DEPARTAMENTO DE INFORMÁTICA

PROYECTO DE FIN DE CARRERA

ASISTENTE MÓVIL PARA ACTUACIÓN EN CASO DE ACCIDENTE DE TRÁFICO

(FIRST AID ASSISTANT)

AUTOR: DAVID CACHO CACHO

TUTOR: JAVIER GARCÍA GUZMÁN

LEGANÉS, OCTUBRE DE 2013

TÍTULO: ASISTENTE MÓVIL PARA ACTUACIÓN EN CASO DE ACCIDENTE DE TRÁFICO (FIRST AID ASSISTANT)

AUTOR: DAVID CACHO CACHO

DIRECTOR: JAVIER GARCÍA GUZMÁN

EL TRIBUNAL

PRESIDENTE:

VOCAL:

SECRETARIO:

Realizado el acto de defensa y lectura del Proyecto de Fin de Carrera el día 30 de Octubre de 2013 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

A todos mis compañeros, especialmente a Aarón y Marcos, que me han acompañado durante todos estos años, compartiendo buenos momentos que hicieron más livianos los largos días y noches de estudiar y hacer prácticas.

A Gema, por su inestimable ayuda revisando la memoria para ayudarme a corregir esas erratas de las que suelo pecar.

A Irene, por estar ahí para azuzarme un poco cuando no había muchas ganas de avanzar con el proyecto y por aguantar las exposiciones preparando la presentación.

A Javier, mi tutor, por darme la oportunidad de llevar a cabo este proyecto y ayudarme con las dudas que me fueron surgiendo durante el desarrollo para poder llegar hasta aquí.

Y como no, a Alfredo y Lourdes, mis padres, que han estado ahí año tras año, me fuese bien o mal. Sin ellos jamás habría podido llegar hasta esta meta.

TABLA DE CONTENIDO

1

INTRODUCCIÓN AL PROYECTO

1. INTRODUCCIÓN AL PROYECTO

1.1. ESTRUCTURA DEL DOCUMENTO

El presente documento tiene como objetivo describir los distintos aspectos relacionados con el desarrollo del Proyecto de Fin de Carrera. Se va a dividir en las siguientes secciones:

1. **Introducción al proyecto**

   En esta sección se proporcionará una introducción al proyecto, indicando principalmente el motivo por el que se ha optado por desarrollarlo y como pretende ayudar a solucionar los problemas mencionados.

2. **Estado del arte**

   En esta sección se explicará la evolución histórica del desarrollo de aplicaciones, hasta llegar al desarrollo móvil, para terminar mostrando una introducción a la programación en Android junto con un ejemplo de aplicación sencillo.

3. **Análisis del sistema**

   En esta sección se abordará el análisis del sistema, mostrando los requisitos que se persigue conseguir con el desarrollo y los casos de uso.

4. **Implementación**

   En esta sección se mostrará como se ha llevado a cabo el diseño y la implementación de la aplicación, de forma detallada. Se incluirán distintos diagramas de secuencia y de datos para completar el conocimiento de la aplicación.

5. **Plan de pruebas**

   En esta sección se mostrará un plan de pruebas definido, para valorar la correcta implantación de todos los requisitos y casos de uso, junto con los resultados obtenidos en las mismas.

6. **Conclusiones**

   Esta sección queda dedicada a las conclusiones obtenidas tras el desarrollo del proyecto, incluyendo posibles mejoras futuras que incluir a posteriori.

Además de los puntos citados, el documento contará con un conjunto de anexos, divididos en:

- **Anexo I: Manual de usuario**

  Este anexo contendrá un manual de uso, con el cual los usuarios podrán saber cómo sacar el máximo provecho de la aplicación.

- **Anexo II: Índice de tablas**

  Este anexo contendrá un índice con las tablas incluidas en el proyecto, de forma que se pueda acceder rápidamente a ellas en caso de necesidad.

- **Anexo III: Índice de figuras**

  Este anexo contendrá un índice con las figuras incluidas en el proyecto, de forma que se pueda acceder rápidamente a ellas en caso de necesidad.

- **Anexo IV: Referencias**

  Este anexo contendrá las referencias utilizadas en distintos apartados del documento, para que puedan ser consultadas en caso de necesidad.

1.2. INTRODUCCIÓN

En pleno auge de la "Era Tecnológica", los *smartphones* copan el primer lugar en cuanto a importancia para los usuarios, seguidos en los últimos años por las *tablets*. Para poder comprender esta importancia, conviene remontarse a los inicios de los llamados teléfonos inteligentes. Mucha gente piensa que todo comienza con la entrada en liza del teléfono *iPhone*, pero la realidad es que antes de la llegada de este terminal, hubo otros que comenzaron a marcar el camino.

Todo comenzó en 1993, de la mano de IBM. Fue entonces cuando apareció en el mercado de teléfonos móviles el llamado *Simon*. Este fue el primer intento real de crear un teléfono "con algo más", mediante la incorporación de servicios de voz y de datos. Además de funcionar como teléfono móvil, era un asistente digital personal, ya que incluía calendario, libreta de direcciones, reloj mundial, calculadora, bloc de notas, correo electrónico, juegos e incluso funcionaba como fax. Lo increíble de este terminal, teniendo en cuenta cuando apareció, es que contaba con una pantalla táctil y un teclado *QWERTY*. Sin embargo, su precio de 900 dólares suponía una pequeña fortuna.

FIGURA 1: SIMON DE IBM

En 1996 surgió la *Palm Pilot*. Aunque técnicamente no puede ser considerado como un s*martphone*, resultó clave para popularizar el uso de terminales móviles, acostumbrando a los usuarios a la idea de poder llevar sus datos de un lado a otro. Gracias a su precio de 300 dólares, que ahora puede parecer mucho pero para la tecnología del momento no era demasiado, fue muy utilizado por ejecutivos y hombres de negocios. Este terminal fue el que popularizó las siglas *PDA* (Personal Digital Assistance).

FIGURA 2: PALM PILOT

En 1998, la empresa Nokia entró en el mercado con su modelo *9110 Communicator*. Su diseño es más parecido a lo que hoy entendemos como *smartphone*, aunque no se permitía la navegación por internet. Su teclado *QWERTY* deslizable sirvió como base para los modelos de teléfonos más actuales.

FIGURA 3: NOKIA 9110 COMMUNICATOR

A principios del siglo XXI, concretamente en 2002, la compañía canadiense RIM entró por la puerta grande en el mercado de los teléfonos móviles con su *BlackBerry 5810*. Se trataba de un teléfono con la capacidad de revisar correos electrónicos y navegar por internet. Su principal escollo fue que para hablar por teléfono era necesario utilizar auriculares, ya que, aunque parezca increíble, el terminal no tenía altavoces. La compañía solucionó este problema dos años después con el lanzamiento de su modelo *BlackBerry 6210*.

FIGURA 4: BLACKBERRY 5810

En 2007 se produjo en verdadero boom de los s*martphone*. En aquel año, Apple mostró su primer *iPhone*, con el que logró entrar en el mercado de la telefonía móvil de una forma espectacular. Este terminal vendió millones de unidades, en gran parte gracias a su pantalla táctil, que ofrecía la mejor experiencia de uso hasta ese momento. Este modelo fue, y sigue siendo en muchos aspectos, la referencia con la que los demás terminales son comparados.

FIGURA 5: IPHONE

El mismo año que aparecía en terminal de Apple, Google presentó su sistema operativo Android. En aquel momento el lanzamiento no causó tanto revuelo, y tardó unos años en consolidarse como un éxito, y como el principal rival de Apple en el mundo de los s*martphone*. Concretamente, en 2009 se lanzó el primer terminal exitoso que utilizaba Android como sistema operativo, el *Droid* de Motorola.

FIGURA 6: DROID DE MOTOROLA

Desde aquel momento, la industria sufrió una expansión enorme, con multitud de dispositivos en el mercado, y con Android ganándo la cuota de ventas a *iPhone*, si bien el primero es un sistema operativo usado por muchos terminales.

La culminación de la revolución producida por los s*martphone* se produjo en 2010, cuando entra en escena la tableta de Apple *iPad*. A partir de aquel momento, las *tablets* entran en escena para quedarse.

Como en toda evolución, siempre hay comparaciones, y a día de hoy, en el mundo de los s*martphone* y las *tablets*, la comparación principal es la llevada a cabo entre Apple y Android. Pese a que en los comienzos eran varias las compañías que copaban los primeros lugares en cuanto a cuota de mercado, desde 2007 han visto mermados sus datos, hasta quedar algunas prácticamente sin cuota de mercado.

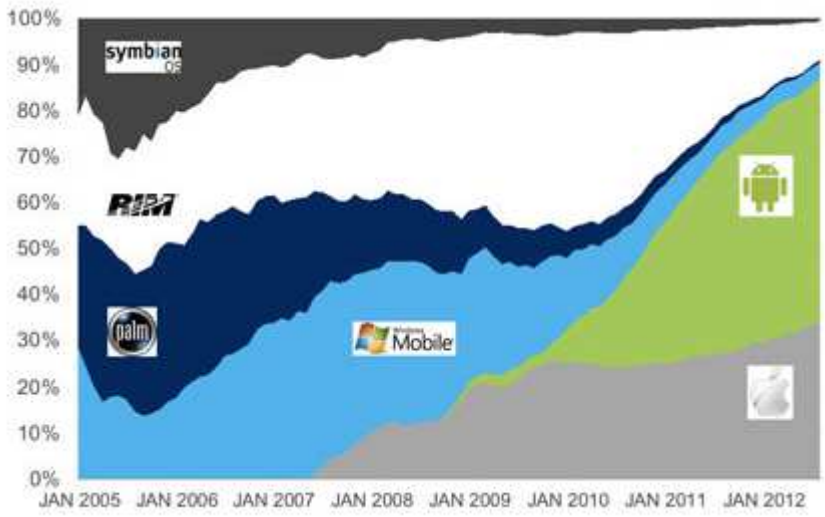


FIGURA 7: EVOLUCIÓN DEL MERCADO

Hasta aquí hemos visto la evolución que han seguido los terminales hasta llegar a lo que tenemos hoy en día, pero esto no nos proporciona el por qué del auge del uso de s*martphone* y *tablets*, que a día de hoy ya han desbancado en cuota de ventas a los ordenadores tradicionales. Para saber esto, debemos conocer las ventajas que aportan estos dispositivos:

- Los s*martphone* se expandieron en forma de *tablet* y estas compiten ahora en tamaño de pantalla, una de las pocas ventajas que les quedan a los ordenadores tradicionales.
- Los ordenadores tradicionales están más orientados a productores de contenido, y la gran mayoría de las personas son consumidoras de contenido. Por ello el ámbito laboral sigue dominado por los ordenadores tradicionales, mientras que el ocio lo copan los s*martphone* y *tablet*.
- Los s*martphone* son mucho más fáciles de llevar a todas partes lo que unido al gran conjunto de aplicaciones, les aportan utilidad en situaciones diferentes (por ejemplo, para hacer deporte).
- Los s*martphone* y *tablet* ofrecen conectividad tanto WiFi como a través de redes móviles, mientras que los ordenadores tradicionales sólo ofrecen WiFi.
- El almacenamiento en la nube comienza a proporcionar alternativas a la capacidad de almacenamiento de los ordenadores tradicionales.

Con toda esta información en la mano, no es de extrañar que el mercado de aplicaciones diseñadas para *smartphone* y *tablet* esté en aumento. Concretamente, Android y Apple copan la mayor parte de la cuota de mercado (43% y 28% respectivamente, como puede verse en la siguiente imagen), por lo que resulta especialmente útil centrarse en cualquiera de ellos a la hora de realizar nuevas aplicaciones.

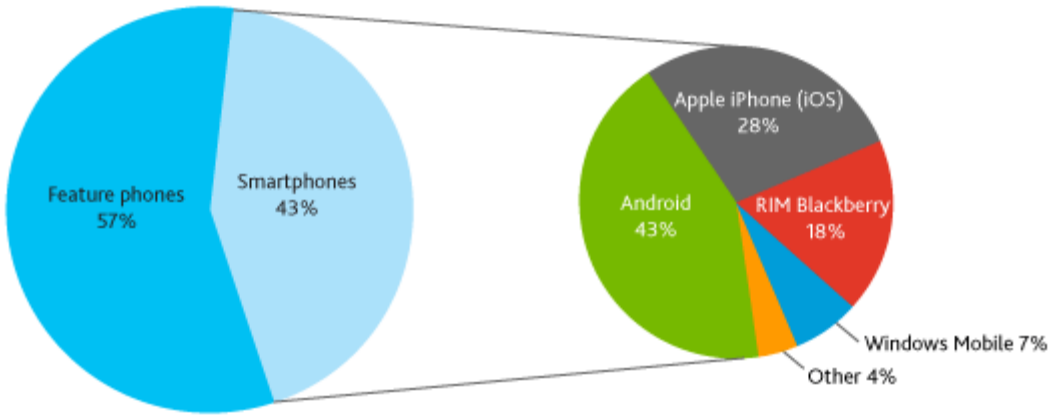


FIGURA 8: COMPARATIVA ACTUAL DEL MERCADO

A la vista de estos datos, se ha decidido desarrollar una aplicación para Android.

1.3. DESCRIPCIÓN DEL PROBLEMA

En nuestro país, según UNESPA (Unión Española de Entidades Aseguradoras y Reaseguradoras) se producen anualmente al menos 2.000.000 de accidentes con daños materiales. Además, en el año 2011, el número de accidentes con víctimas informados por la policía fue de 83.027. Estos datos reflejan la importancia de estos hechos en nuestro país.

Además de intentar reducir el número de accidentes que se producen a lo largo de un año, cobra mucha importancia el conocer como se ha de actuar en el momento de sufrir o ver un accidente de tráfico. Los primeros minutos pueden ser muy importantes, y conocer el patrón de actuación en casos de emergencia puede suponer una gran diferencia a la hora de ayudar a las víctimas de un accidente.

La formación de los usuarios resulta de vital importancia para optimizar la atención y los tiempos de respuesta tras un accidente. Según datos proporcionados por RACE, el 75% de los fallecimientos se producen en la conocida como “Hora de Oro”,

correspondiente a los siguientes 60 minutos tras el accidente. Es en esta hora en la que se debe trabajar para reducir el número de fallecimientos, y un correcto protocolo de acción, no solo por parte de los servicios de emergencias, como bomberos, ambulancias, agentes de tráfico…, sino también por parte de los usuarios, puede poner en marcha todo el mecanismo de auxilio de una forma mucho más eficiente.

El proyecto que se va a desarrollar intenta enfocar la tesitura en la que se pueden encontrar los usuarios ante la perspectiva de tener que afrontar un accidente, ya sea propio o ajeno, para que sean capaces de tomar las decisiones oportunas según la situación. Es importante señalar que un usuario sin conocimientos de primeros auxilios no va a ser capaz de proporcionar la misma ayuda que los servicios de emergencias, pero con su ayuda, puede reducir los riesgos de que se produzcan mayores problemas, y facilitar la labor de los expertos cuando lleguen al lugar del accidente.

1.4. APORTES DEL PROYECTO AL PROBLEMA

Se pretende desarrollar una aplicación que pretenda servir de guía a los usuarios para actuar de forma rápida y coherente ante un accidente de tráfico, facilitando el trabajo de los servicios de emergencias.

Como ha se ha mendionado, los *smartphone* se han convertido en un elemento prácticamente esencial en nuestras vidas, y su expansión es tal que muchos de los usuarios de teléfonos móviles cuenta ya con un terminal de este tipo. Y cabe esperar que en el futuro, el número de personas con este tipo de teléfonos siga creciendo. Por ello, desarrollar una aplicación móvil que guíe a los usuarios para saber cómo actuar en los accidentes de tráfico, hará que las pautas mostradas en ella estén al alcance de una gran cantidad de personas.

Con esta aplicación no se pretende formar a las personas como expertos en primeros auxilios, sino darles la oportunidad de ayudar. Para ello, se va buscar que la aplicación cumpla una serie de requisitos de usabilidad:

- La aplicación debe ser intuitiva y con una interfaz que facilite su uso, ya que se prevé que sea utilizada en situaciones que pueden considerarse de tensión y problemáticas.
- Se mostrarán los pasos a seguir en caso de que el usuario haya visto un accidente, mostrando las distintas acciones que deberá realizar de forma

consecutiva. Estas acciones se verán apoyados por imágenes y audios, para que la información quede lo más clara posible.

- Se mostrarán los pasos a seguir en caso de que el usuario haya sufrido un accidente. Este punto está orientado a accidentes leves, ya que en caso de que se produzca uno grave, es poco probable que se esté en condiciones de utilizar el teléfono.
- Se mostrará la información de la localización del accidente, para poder reportarlo a las unidades de emergencias, así como se facilitará un acceso rápido para llamar al teléfono de emergencias.
- Se permitirá grabar los datos básicos de los accidentes, de forma que puedan ser consultados posteriormente.

Una vez finalizada la aplicación, esta será distribuida gratuitamente para los dispositivos Android, de forma que pueda ser útil y accesible para los usuarios.

2

ESTADO DEL ARTE

2. ESTADO DEL ARTE

2.1. HISTORIA DEL DESARROLLO SOFTWARE

La raíz del desarrollo software se remonta a las décadas de 1960 y 1970. Fue entonces, cuando se hicieron las primeras contribuciones destacadas a través de investigaciones, en las cuales se llegó a la conclusión de que el desarrollo software se trata de un proceso complejo, que involucra varios factores. Los investigadores se percataron de que el desarrollo software no trataba sólo de crear lenguajes de programación y herramientas efectivas, sino que se trataba de un esfuerzo conjunto, complejo y creativo. Debido a esto, vieron que el desarrollo software estaba fuertemente ligado a las personas que participaran en el mismo, su organización y los procedimientos usados para llevar a cabo la tarea.

Esta idea, durante esas décadas, hizo que los investigadores centrasen sus esfuerzos en tres objetivos principales:

- Desarrollo de lenguajes de programación estructurados
- Desarrollo de métodos y principios de diseño
- Definición de ciclos de vida del software

Conviene centrarse especialmente en el tercero de estos objetivos.

Los ciclos de vida determinan las diferentes fases o etapas por las que debe pasar cualquier proceso software, tanto durante su desarrollo, como una vez finalizado el mismo. Normalmente, tenemos un conjunto de fases que son comunes a todos los ciclos de vida. Estas fases son:

- Análisis y especificación de requisitos
- Diseño
- Desarrollo
- Validación
- Despliegue del software
- Mantenimiento
- Retiro

Cada uno de los ciclos de vida define más concretamente qué se debe hacer en cada una de las fases, y lo que es más importante, cuándo han de hacerse. Esto incluye el momento en el que realizar la transición entre distintas fases. Concretamente, las transiciones de una fase a otra deben responder a dos cuestiones básicas: *¿Qué debo hacer a continuación? ¿Durante cuánto tiempo debo seguir haciéndolo*?

Vamos a comentar brevemente los distintos ciclos de vida que se han ido utilizando a lo largo de la historia, para tener una visión global de cómo ha sido la evolución del desarrollo software.

- Prueba y error

Se trata del modelo básico utilizado en los comienzos del desarrollo software. Consiste en escribir código, probarlo, y solucionar los errores que vayan surgiendo en el código.

Este modelo tienes tres problemas principales:

- Tras varios arreglos, el código comienza a volverse pobremente estructurado, con el consiguiente problema de coste para reparar futuros errores a medida que el software crezca.
- Lo normal es que los requisitos finales difieran de los iniciales, ya que los clientes a menudo no saben concretamente lo que quieren, y suelen cambiar cosas. Esos cambios serán muy complicados de contemplar con este modelo.
- No se dispone de un programa de pruebas con el que validar el software.

- Modelo en cascada

Este modelo se nutrió de los problemas derivados de la forma de trabajar en el desarrollo del software, para definir una serie de fases consecutivas que deberían llevarse a cabo. Estas fases pueden verse en la siguiente figura:

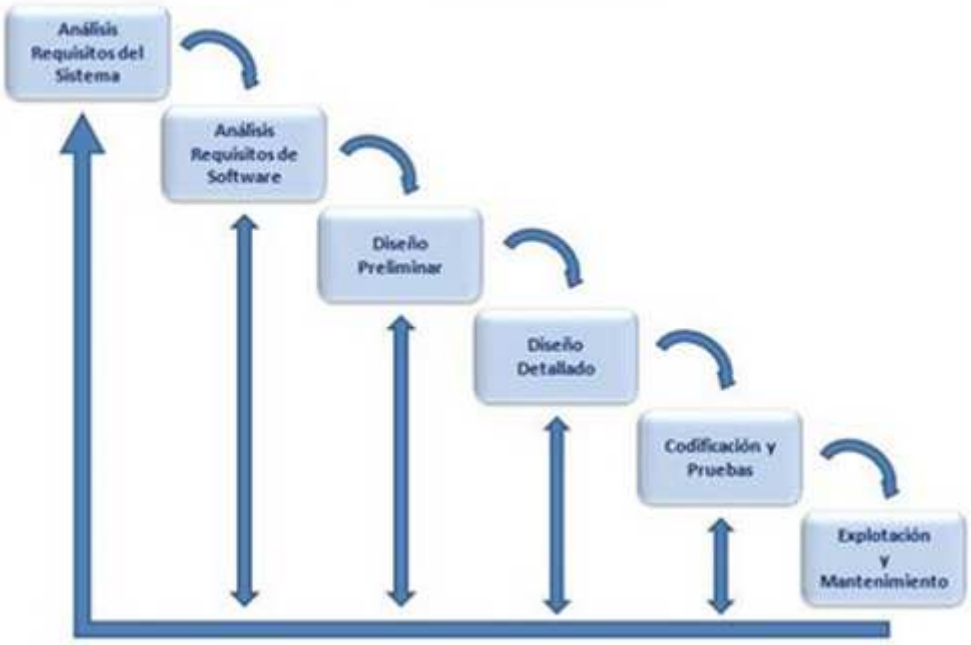


FIGURA 9: MODELO EN CASCADA

Frente a la forma tradicional de desarrollar software, el modelo en cascada proporcionó un marco de trabajo identificando pasos concretos y posibilidad de volver atrás en alguna de las etapas, si fuese necesario, dotando al desarrollo software de una gran versatilidad.

En modelo en cascada enseguida se convirtió en el ciclo básico de desarrollo para cualquier producto software. No obstante, el modelo no era perfecto, pero a partir de modificaciones incluidas con el paso del tiempo, se fue perfeccionando hasta ser el modelo de referencia, incluso llegando a usarse en la actualidad.

Pese a todo, el modelo en cascada también ofrece problemas. Este modelo funciona muy bien en el desarrollo de software que requiere un gran análisis para su diseño. Sin embargo, para software orientado a un aspecto más interactivo, que requieren menos análisis y más evolución a partir de las pruebas, supone un obstáculo.

- Desarrollo evolutivo

El desarrollo evolutivo nace como consecuencia de las limitaciones del modelo en cascada para hacer frente de forma eficiente a desarrollos software en los que es necesario ir introduciendo cambios a medida que el desarrollo avanza. Este modelo

responde al requisito de los clientes expresado como *"No puedo decirte lo que quiero, pero lo sabré cuando lo vea"*.

El modelo consiste en partir de un prototipo básico, que será evaluado por el cliente, quién decidirá que quiere a continuación. Este modelo puede asemejarse al modelo basado en prueba y error, pero ofrece las ventajas de definir distintos estados para cada ciclo de desarrollo entre evaluaciones del cliente. Sin embargo, esto también tiene consecuencias negativas. A medida que se introduzcan cambios por deseo del cliente, el software va a ser más caótico, y puede llegar un momento que no sea sencillo modificar cosas por la alta cohesión con otros aspectos desarrollados a posteriori.

- Modelo en espiral

El modelo en espiral es un compendio de los modelos explicados anteriormente, en el que se pretende aglutinar los beneficios de todos ellos, minimizando sus desventajas. El ciclo de vida en este modelo, se puede observar en la siguiente figura:

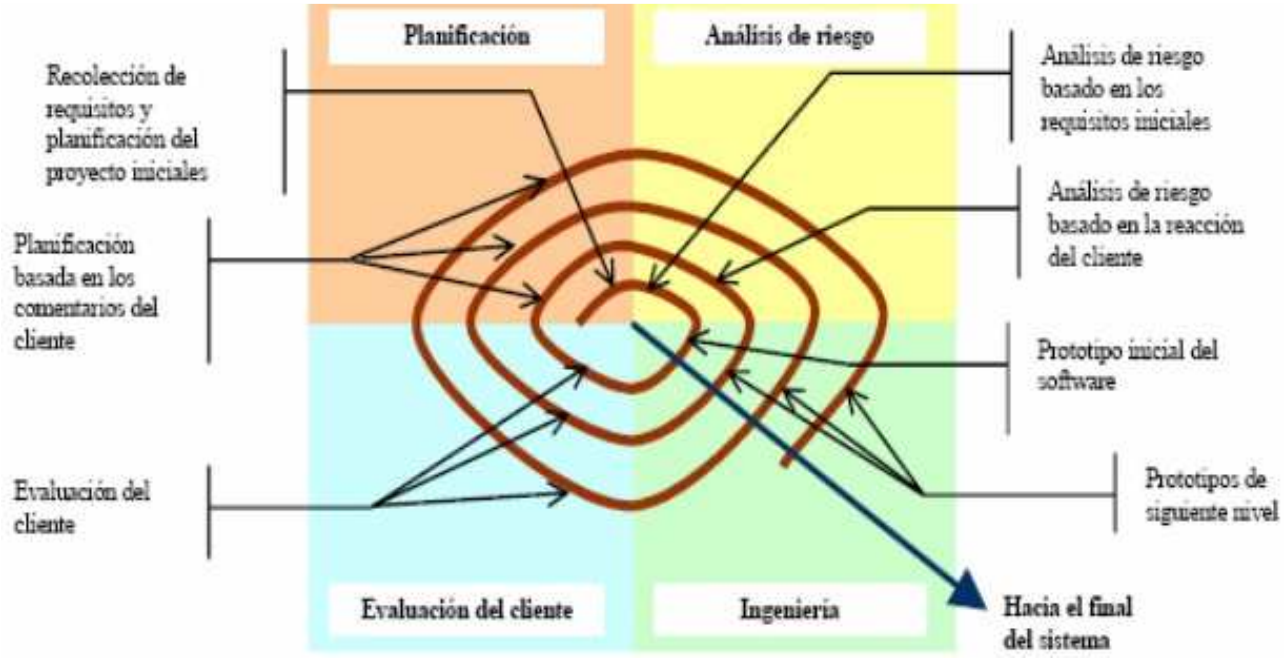


FIGURA 10: MODELO EN ESPIRAL

Como puede verse, hay cuatro fases bien diferenciadas, similares a las definidas en el modelo de cascada, con la diferencia de que los objetivos de las fases no son siempre los mismos, sino que dependen del giro en la espiral en el que estemos. De esta forma, se flexibiliza el modelo, para adecuar cada giro a la evolución del software. Además, incorporamos los prototipos al final de cada giro. Estos prototipos actúan de forma

similar a las entregas intermedias de las que constaba el modelo evolutivo, para que el cliente vaya dando su visión del desarrollo, de forma que se puedan anticipar cambios y adaptar el desarrollo ante imprevistos.

- Metodologías ágiles

Se trata de metodologías recientes, que pretenden cambiar por completo la visión de los desarrollos software. Mientras que los modelos tradicionales se centran en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán, las metodologías ágiles dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas.

El punto de partida de estas metodologías es un documento denominado *"Manifiesto Ágil"*, que resume la filosofía *"ágil"*. Según el manifiesto se valora:

- Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas.
- Desarrollar software que funcione más que conseguir una buena documentación.
- La colaboración con el cliente más que la negociación de un contrato.
- Responder a los cambios más que seguir estrictamente un plan.

Estos valores inspiran los doce principios del manifiesto, que son los que diferencia un proceso ágil de otro tradicional. Los dos primeros son generales y resumen el espíritu de esta metodología, mientras que los restantes son las metas y la organización:

I. La prioridad es satisfacer al cliente mediante tempranas y continuas entregas de software que le aporte un valor.

II. Dar la bienvenida a los cambios. Se capturan los cambios para que el cliente tenga una ventaja competitiva.

III. Entregar frecuentemente software que funcione desde un par de semanas a un par de meses, con el menor intervalo de tiempo posible entre entregas.

IV. La gente del negocio y los desarrolladores deben trabajar juntos a lo largo del proyecto.

V. Construir el proyecto en torno a individuos motivados. Darles el entorno y el apoyo que necesitan y confiar en ellos para conseguir finalizar el trabajo.

VI. El diálogo cara a cara es el método más eficiente y efectivo para comunicar información dentro de un equipo de desarrollo.

VII. El software que funciona es la medida principal de progreso.

VIII. Los procesos ágiles promueven un desarrollo sostenible. Los promotores, desarrolladores y usuarios deberían ser capaces de mantener una paz constante.

IX. La atención continua a la calidad técnica y al buen diseño mejora la agilidad.

X. La simplicidad es esencial.

XI. Las mejores arquitecturas, requisitos y diseños surgen de los equipos organizados por sí mismos.

XII. En intervalos regulares, el equipo reflexiona respecto a cómo llegar a ser más efectivo, y según esto ajusta su comportamiento.

A continuación podemos ver una tabla resumen con las diferencias entre las metodologías tradicionales y las metodologías ágiles:

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

TABLA 1: DIFERENCIAS ENTRE METODOLOGÍAS ÁGILES Y TRADICIONALES

2.2. DESARROLLO MÓVIL

Con la llegada de los *smartphone* y las *tablets*, se ha producido un cambio en la forma de desarrollar aplicaciones móviles. Antiguamente, la mayoría de los desarrollos estaban controlados por las grandes compañías que manufacturaban los teléfonos móviles. Ahora, el desarrollo de aplicaciones móviles ha dado un vuelco, pasando a ser los desarrolladores independientes los que crean la gran mayoría de las aplicaciones móviles. Y no solo cambia quién desarrolla las aplicaciones, sino también la forma de hacerlas.

A lo largo de la historia del desarrollo software, las metodologías de trabajo han ido evolucionando, y ahora, con el desarrollo móvil, se ha dado una nueva vuelta de tuerca. En muchos aspectos, el desarrollo de aplicaciones móviles es similar a la ingeniería de software para las aplicaciones corrientes. Sin embargo, las aplicaciones móviles presentan requisitos adicionales que son menos comunes en las aplicaciones tradicionales. Entre estos requisitos podemos destacar los siguientes:

- Potencial interacción con otras aplicaciones: La mayoría de las aplicaciones tradicionales únicamente dependen del software instalado. En el caso de las aplicaciones móviles, estas pueden venir de distintas fuentes que interactúen entre ellas.
- Sensores: Los *smartphone* y las *tablet* incorporan sensores de distintos tipos, lo que dota a las aplicaciones de posibilidades no contempladas en las aplicaciones tradicionales.
- Seguridad: Los sistemas móviles son abiertos, por lo que es fácil instalar aplicaciones malignas que puedan afectar al funcionamiento del dispositivo, incluso transmisión de datos locales.
- Interfaces de usuario: En una aplicación móvil, las posibilidades que tenemos de pantalla son más limitadas, y la forma de utilizar varía sensiblemente con respecto a la utilización de las aplicaciones tradicionales.
- Consumo de energía: Muchos de los aspectos de una aplicación afectan al consumo de energía del dispositivo, lo que hace que los dispositivos móviles sean especialmente vulnerables debido a sus baterías.

Como resumen de estos puntos, podemos indicar que una de las grandes metas del desarrollo móvil se centra en la experiencia del usuario. Cualquier aplicación móvil

debe girar en torno a los sensores. La pantalla táctil será el lugar predominante de entrada y salida de la información, por lo que se debe hacer un especial hincapié en dar al usuario una forma cómoda de moverse por ella. Los gestos y los *widgets* son dos opciones que ayudaran en este aspecto. Al fin y al cabo, las aplicaciones móviles dejan de lado la tradicional navegación de ventanas, iconos, menús y punteros, por lo que se debe de proporcionar buenos sustitutos. Y todo esto recudiendo considerablemente el tamaño de ventana. Se terminó el disponer de pantallas con muchas cosas en ellas. Lo que se muestre debe ser claro y conciso.

Debemos tener en cuenta que no toda la experiencia de usuario se basa únicamente en la pantalla. Los requisitos no funcionales de la aplicación tienen mucha importancia. Entre ellos debemos destacar el uso eficiente de los recursos del dispositivo, la robustez de la aplicación (bloquear un dispositivo móvil puede ser desastroso para la opinión del usuario) y la seguridad.

Estas son solo algunas pequeñas pinceladas que hay que tener en cuenta a la hora de analizar una aplicación móvil. Lo cierto es que tras cientos de miles de aplicaciones móviles desarrolladas, no hay ninguna solución formal que indique que proceso hay que seguir durante el desarrollo. Muchos de los desarrolladores, incluidos los desarrolladores individuales, siguen una metodología ágil, basada en sucesivas versiones que son liberadas para que el usuario pueda actualizar la aplicación.

Para llevar a cabo la creación de las aplicaciones, los desarrolladores cuentan con entornos de desarrollo muy completos, que les permiten hacer uso de gran cantidad de instrumentos incluidos en el dispositivo. Una vez desarrollan la aplicación, disponen de un portal en el cual pueden poner a disposición de todos los usuarios sus aplicaciones para que se las descarguen. Este modelo, que se puede observar en la siguiente figura, cambia sustancialmente la forma de desarrollar aplicaciones móviles, incluyendo a los desarrolladores, los fabricantes con los portales de contenidos y los consumidores finales.

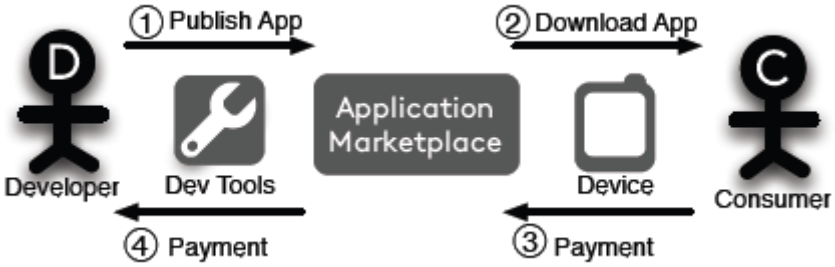


FIGURA 11: MODELO DE DESARROLLO DE APLICACIONES MÓVILES

Para comprender mejor este cambio en el desarrollo de aplicaciones móviles, vamos a centrarnos en distintos aspectos, para verlos más en profundidad.

- Herramientas de desarrollo

En el núcleo de toda plataforma móvil se encuentra su SDK (*Software Development Kit*). Este permite a los desarrolladores acceder a gran cantidad de librerías y emuladores, para hacer uso de todos los elementos disponibles en el dispositivo, de manera que puedan asegurarse de que la aplicación a desarrollar funcione correctamente en los dispositivos finales de los usuarios. No obstante, la forma de compartir ese SDK con los desarrolladores varía considerablemente de unas compañías a otras. Algunas deciden restringir el acceso, mientras que otras prefieren difundir la totalidad de su SDK y su sistema operativo. En función de cómo se comporten, podemos hacer dos diferenciaciones: las que utilizan el modelo "catedral" y las que utilizan el modelo "bazar".

- *El modelo "catedral"*

Con este modelo, las empresas propietarias mantienen su SDK y su sistema operativo oculto para aquellos externos que no realicen el pago correspondiente por su uso. Algunas de las compañías que siguen este modelo son Apple y RIM. Este modelo ofrece como ventaja el poder controlar de forma más seguro que se desarrolla para tu plataforma.

- *El modelo "bazar"*

En contraste con el modelo "catedral", este modelo permite a los desarrolladores acceder a todo el código original del SDK y de su sistema operativo. Este tipo de modelo fue la base de Linux, y ahora Android continua dándole uso. Los beneficios que se obtienen con el uso de este modelo se centran en la reducción de costes de desarrollo

y mantenimiento de la plataforma, lo que conlleva a una reducción en los costes finales, posibilitando que se incremente el número de consumidores.

- Portales

Para poder permitir que las aplicaciones desarrolladas lleguen a los clientes finales, se crean los portales de aplicaciones. Estos portales son esenciales en la distribución de aplicaciones móviles, ya que juegan el papel de intermediario entre los desarrolladores y los clientes. Algunas empresas optan por utilizar portales centralizados como estrategia de ventas, mientras que otras prefieren optar por una opción descentralizada, para tener múltiples puntos de ventas.

- *Portal centralizado*

En este modelo, un único portal es propuesto como portal principal de descargas, dónde la mayoría de las aplicaciones son publicadas. Esta opción da al portal una ventaja competitiva sobre el resto de opciones para descargar aplicaciones. Sin embargo, incluso dentro este modelo, podemos encontrar dos diferentes formas de llevarlo a cabo. Apple empuja a los desarrolladores a utilizar un único y exclusivo portal, con un estricto proceso de revisión de aplicaciones. Esto hace que los desarrolladores busquen alternativas en portales “negros”, como por ejemplo Cydia. Google por otro lado, no restringe las publicaciones en su portal y no posee ningún plan de revisión de aplicaciones antes de su publicación.

Este tipo de portales ofrecen a los desarrolladores la posibilidad de aliviarse de tener que controlar varios aspectos relativos a la distribución de aplicaciones, como es la facturación y la publicidad de las aplicaciones. Estos puntos son tratados por el portal. Antes de la existencia de este tipo de portal, los desarrolladores tenían que poner en manos de aplicaciones de terceras personas la responsabilidad de efectuar los cobros de las descargas. Sin embargo, no todo es perfecto con los portales centralizados. Como punto negativo está el hecho de que los desarrolladores deben seguir ciertas reglas definidas por la plataforma que controla el portal.

- *Portal descentralizado*

Con esta opción, los desarrolladores tienen libertad para publicar sus aplicaciones en cualquier portal de terceros. Esto hace que los portales compitan entre ellos para ganar aplicaciones y clientes. El problema de este modelo, es la dificultad que pueden tener

los clientes para encontrar aplicaciones entre toda la variedad de portales, perdiendo las aplicaciones la posibilidad de llegar a todos los usuarios.

- Integración

Algunas plataformas centran sus esfuerzos en sus centros de negocio, los cuales proporcionan un sistema operativo con centros de soporte para los desarrolladores, mientras que otras integran todo el proceso de distribución. Debido a esto, podemos clasificar a las empresas según su nivel de integración. A continuación podemos ver una imagen con los distintos tipos de integración.

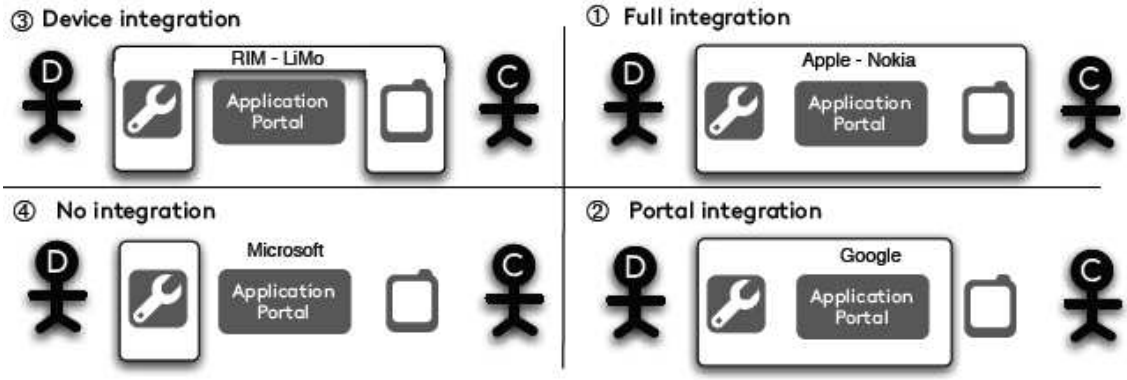


FIGURA 12: MODELO DE INTEGRACIÓN DE APLICACIONES MÓVILES

- *Integración completa*

Las plataformas con una integración completa tienen un estricto control sobre cada paso en el proceso de distribución, desde la creación de dispositivos hasta la publicación de aplicaciones.

- *Integración en portal*

Las plataformas que ofrecen una integración basada en un portal, centran su atención en el desarrollo de aplicaciones y su distribución mediante la integración de un portal. Este tipo de plataformas se independizan de la creación de dispositivos que utilizan sus sistemas.

- *Integración de dispositivos*

En la integración con dispositivos, las plataformas además de crear las herramientas de desarrollo, crean los dispositivos que las van a utilizar. Sin embargo, en esta integración, quedan al margen del portal de distribución de aplicaciones.

- *Sin integración*

Finalmente, tenemos algunas plataformas que no tienen ningún tipo de integración. Esto quiere decir que son plataformas centradas en desarrollar su sistema operativo y las herramientas para utilizarlo, pero se quedan al margen tanto del desarrollo y publicación de aplicaciones, como de la creación de dispositivos que utilicen su plataforma.

➢ Tendencias

Tras ver detenidamente todos los aspectos que envuelven el desarrollo de aplicaciones móviles, tenemos que destacar que las plataformas no son estáticas, y algunos métodos son tomados como referencia y copiados de unas plataformas a otras. Esto hace que la industria vaya evolucionando. Conviene fijarnos detenidamente como se han ido moviendo las principales plataformas, para poder entender hacia dónde tiende el desarrollo.

Antes de la llegada de Apple y su *AppStore*, y más recientemente con el *Android Market* de Google, las plataformas basaban su negocio en portales descentralizados. Sin embargo, Apple demostró que el mercado de las aplicaciones móviles no debe ser subestimado. Tras ver el enorme efecto que produjo el *AppStore* y sus resultados en corto tiempo, las plataformas tradicionales comenzaron a moverse en dirección a los portales centralizados, abandonando sus antiguas creencias.

FIGURA 13: TENDENCIA EN EL USO DE PORTALES DE APLICACIONES

En el campo de las tecnologías utilizadas, la gran mayoría de las plataformas móviles utilizaban tecnologías cerradas, y únicamente LiMo utilizaba código abierto. Sin embargo, Nokia tomó conciencia de las posibilidades del código abierto cuando adquirió el sistema operativo Symbian. Con la llegada de Google, quién también utiliza código abierto, se ha abierto una batalla para ver que es mejor, si las tecnologías abiertas, con Google, LiMo y Nokia, y las tecnologías cerradas, propias de Apple, RIM y Microsoft. En esta batalla, las tecnologías abiertas parten con las posibilidades de

reducir los costes de desarrollo e incrementar el número de consumidores. Un mayor número de consumidores implica un mayor número de clientes para las aplicaciones, lo que lleva a mayores posibilidades para desarrollar aplicaciones. Por otro lado, una tecnología abierta, no puede ofrecer el mismo soporte que una tecnología cerrada.

La última tendencia a considerar gira en el ámbito de la integración de las plataformas. Antes de la llegada de Apple, no había ninguna plataforma que contara con una integración completa. Es más, tampoco existía ninguna plataforma con una integración en portal hasta la llegada de Google al mundo de los *smartphone*. Con la llegada de ambos, las plataformas estudiaron todas las posibilidades, de forma que fueron migrando su forma de trabajar, hasta llegar al mapa de integración que tenemos actualmente.

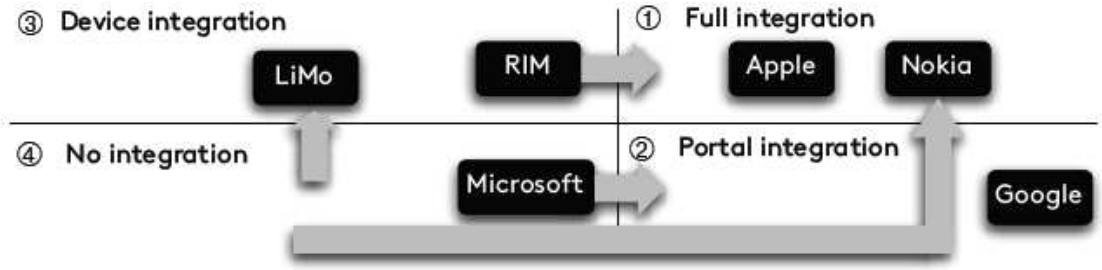


FIGURA 14: TENDENCIA EN LA INTEGRACIÓN DE APLICACIONES MÓVILES

Nokia, tras adquirir el sistema operativo Symbian, pasó de no contar con ningún tipo de integración, a tener una integración completa. RIM, por su parte, está cambiando su tendencia hacia la integración completa, pero en su caso desde la integración de dispositivos. Finalmente, Microsoft está intentando no quedarse atrás, para lo cual está empezando a implementar una integración en portal, compitiendo así en igualdad de oportunidades que Google.

2.3. INTRODUCCIÓN A ANDROID

En esta sección hablaremos de los primeros pasos que hay que seguir para poder desarrollar cualquier aplicación para un dispositivo móvil Android.

- Entorno de desarrollo

A la hora de llevar a cabo cualquier tipo de desarrollo, es necesario contar con un grupo de programas que nos permitan realizar la tarea de forma sencilla. Dentro del software necesario para desarrollar una aplicación para Android, debemos tener en

cuenta que está basado en el lenguaje de programación Java. Por lo tanto, necesitaremos algunas herramientas comunes a este lenguaje.

Lo primero que necesitamos para desarrollar cualquier aplicación para Android, es tener instalado la versión del JDK (Java Development Kit) de Java. Este se puede descargar desde la página oficial de Oracle. Simplemente tendremos que escoger la versión que queramos e instalarla como cualquier otro programa.

Una vez tengamos el núcleo en el cual se basa Android, conviene instalar un entorno de desarrollo o IDE. Prácticamente cualquier IDE disponible contará con sus *plugins* correspondientes para desarrollar en Android. Para el desarrollo de este proyecto, se ha optado por trabajar con Eclipse. Para instalar este IDE, basta con ir a la página oficial de Eclipse y seleccionar la versión que queramos descargar. La instalación consiste en descomprimir el archivo *zip* que habremos descargado en la ubicación que nosotros queramos. No hay que realizar ninguna otra acción. Al descomprimir los archivos, en la ubicación seleccionada encontraremos un archivo *eclipse.exe* que lanzará el IDE para comenzar a trabajar con él. La primera vez que arranquemos Eclipse, nos preguntará cual va a ser nuestro espacio de trabajo. Esto no es más que una carpeta en la cual se almacenarán los proyectos que vayamos desarrollando con el IDE. Una vez seleccionado el directorio de nuestro espacio de trabajo, podemos seleccionar la opción para que se utilice dicho espacio como predeterminado, como se muestra en la siguiente imagen.

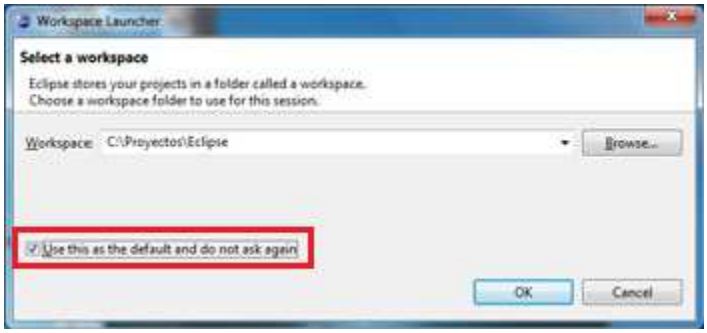


FIGURA 15: CREACIÓN DE UN WORKSPACE EN ECLIPSE

Una vez tengamos nuestro entorno de trabajo listo, es hora de empezar a preparar los requisitos para trabajar con Android propiamente. Lo primero que necesitamos para ello es el entorno de desarrollo de Android, es decir, su SDK. Este se puede descargar de la página oficial de Android. Google proporciona una opción para descargar un paquete

que contiene Eclipse, el SDK y varios componentes ya instalados. Sin embargo, se recomienda instalar las cosas por separado, para disponer de una versión limpia del IDE, por si queremos utilizarlo para más cosas. Dado que en nuestro caso ya tenemos instalado Eclipse, habrá que escoger la versión correspondiente al apartado *"Use an existing IDE"*. Una vez descargado, bastará con instalarlo normalmente.

Tras instalar el SDK de Android, necesitamos instalar un *plugin* en Eclipse que permita utilizar dicho SDK de forma sencilla. Para ello, es necesario descargar el *plugin* ADT (Android Development Tools). Para ello, hay que acceder al menú *Help* → *Install new software…* de Eclipse. En la pantalla que se nos abrirá, seleccionados *Add…* e introducimos lo siguiente:

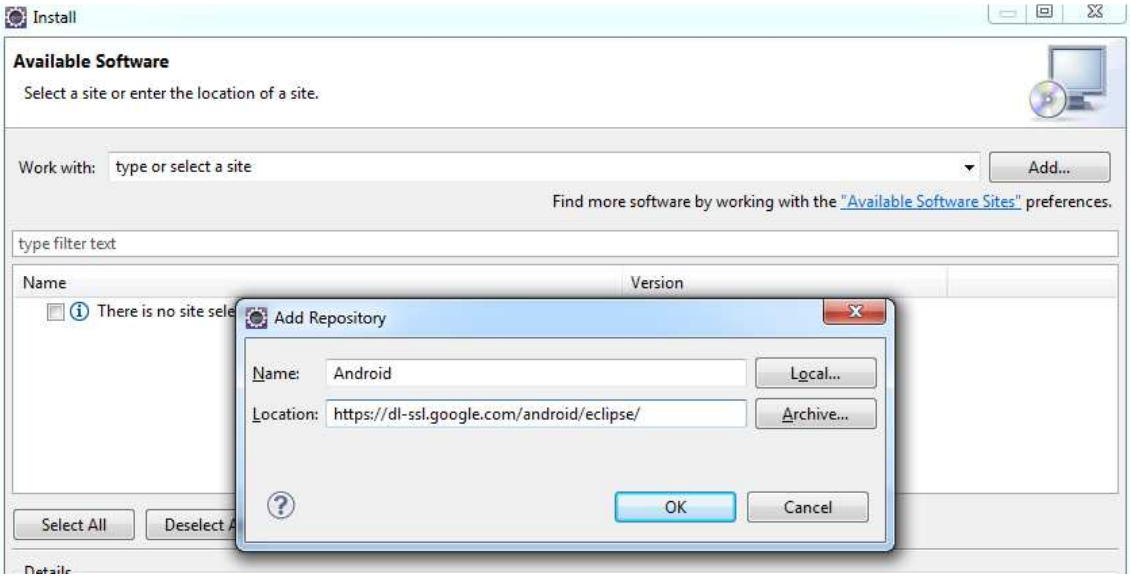


FIGURA 16: INCLUIR EL REPOSITORIO DE ANDROID A ECLIPSE

Tras clickar en *OK* y luego en *Next*, nos aparecerá otra pantalla donde seleccionaremos los dos paquetes disponibles, *Developer Tools* y *NDK Plugins* y tras pulsar *Next* comenzará la instalación. Durante este proceso, se pedirá que aceptes la licencia, y quizás aparezca algún *warning*. En esos casos basta con aceptar para continuar con el proceso de instalación, hasta que finalice.

Cuando haya terminado la instalación y hallamos reiniciado Eclipse, tendremos que configurar el *plugin* recién instalado. Para ello, en el menú *Windows* → *Preferences…* seleccionaremos de entre las opciones que se nos muestran a la izquierda *Android* e indicaremos la ruta en la que instalamos el SDK. Si sale algún *warning* se aceptarán, ya que se solucionarán más adelante.

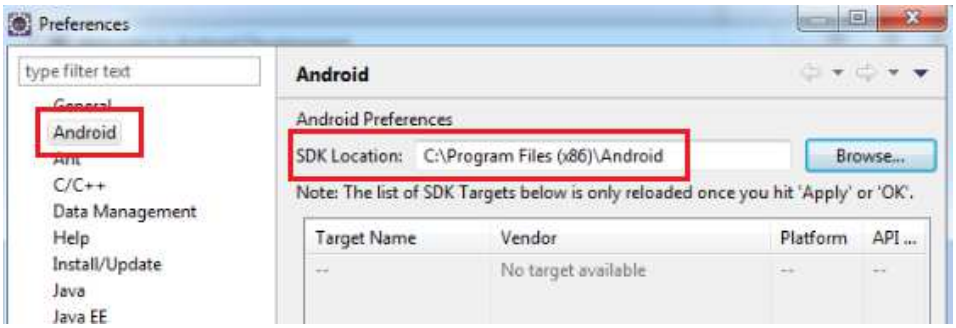


FIGURA 17: CONFIGURAR EN EL ENTORNO DE DESARROLLO DE ANDROID EN ECLIPSE

Ya casi tenemos nuestro IDE listo para trabajar. Cuando descargamos el *plugin* de Android, descargamos dos paquetes, el necesario para utilizar el SDK que ya hemos configurado, y otro con herramientas. Este último nos permite configurar las características básicas de Android sobre las que vamos a trabajar, es decir, la versión sobre la que vamos a desarrollar nuestra aplicación. Lo normal es configurar al menos dos versiones, la que vamos a utilizar como base del desarrollo, y la mínima versión que queremos que soporte nuestra aplicación. Lo normal es utilizar como versión de desarrollo la última disponible, y como versión mínima la 2.2 o 2.3 ya que son versiones muy utilizadas en dispositivos antiguos.

Para seleccionar las versiones con las que se trabajará, se selecciona el menú *Window* → *Android SDK Manager*. Aparecerá una lista con todas las versiones de Android disponibles. En dicha lista se seleccionarán las opciones *Android SDK Platform-tools* y las dos versiones que queramos utilizar. Una vez seleccionado todo, se pulsa el botón de instalación y se espera a que finalice la descarga.

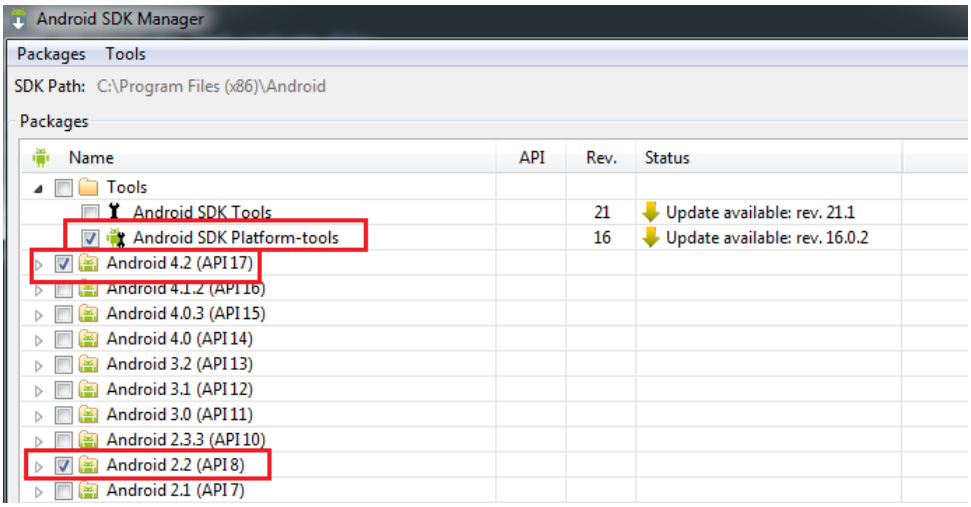


FIGURA 18: INSTALAR VERSIÓN DE ANDROID

Para terminar de dejar nuestro entorno de desarrollo perfecto, podemos instalar simuladores que hagan las veces de dispositivos, para ver como se verá nuestra aplicación. Esto no es necesario si disponemos de un dispositivo real y lo conectamos al ordenador, pero es recomendable, ya que nos permitirá probar la aplicación simulando distintos dispositivos con distintas características. Estos dispositivos simulados se denominan AVD (Android Virtual Device).

Para añadir AVD, se selecciona *Window → AVD Manager* y en la sección *Virtual Devices* se podrán añadir todos los AVD que queramos. Se recomienda instalar al menos dos, uno para la versión mínima que escogimos, y otro para la versión más reciente. Para configurar el AVD basta con darle un nombre descriptivo, la versión de Android que utilizará y el dispositivo de ejemplo que simularemos según su resolución de pantalla.

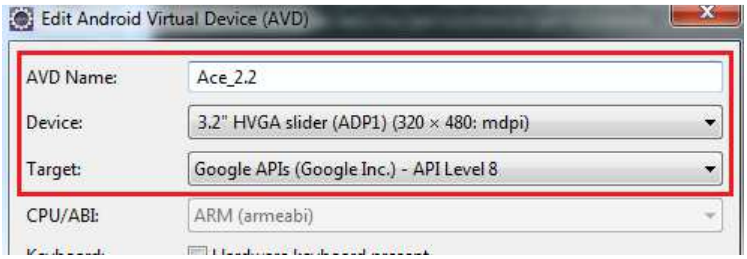


FIGURA 19: CREACIÓN DE UN AVD EN ECLIPSE

- Estructura del proyecto

Crear un nuevo proyecto Android es muy sencillo con Eclipse. Seleccionaremos *File → New → Android Application Project* e iremos rellenando las distintas pantallas que vayamos teniendo.

En la primera simplemente tendremos que indicar el nombre de la aplicación, del proyecto y el del paquete principal. Además seleccionaremos la versión mínima que debe soportar la aplicación y la versión sobre la que desarrollaremos (la última disponible), que coincidirá con la versión de compilación. En la siguiente pantalla nos preguntará si queremos configurar varias opciones. No hace falta modificar ninguna de ellas, por lo que se dejarán seleccionadas las que vienen por defecto. En la tercera pantalla podremos configurar el icono de la aplicación, dónde se podrá seleccionar una imagen predefinido. Tras esto, se nos preguntará por el tipo de *actividad* principal de la aplicación. De momento diremos que una *actividad* es una "pantalla" o "ventana" de la

aplicación. Lo normal es utilizar los valores por defecto. Finalmente, en el último paso, nos pedirá los datos de la actividad principal. Se deberá escoger un nombre para la actividad y otro para su *layout*. Este *layout* será el fichero *xml* que definirá la interfaz gráfica de la actividad.

Una vez finalizado el proceso, nos aparecerá el nuevo proyecto a la izquierda de Eclipse, con una estructura similar a la siguiente:

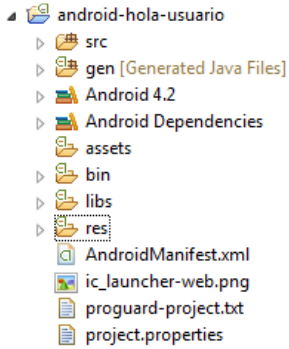


FIGURA 20: ESTRUCTURA DE UN PROYECTO ANDROID

Vamos a ir viendo poco a poco que significa cada uno de los apartados que vemos en la imagen:

- *Carpeta src*

Esta carpeta contendrá todo el código fuente de la aplicación. A la hora de crear el proyecto, Eclipse creará una clase correspondiente a la actividad principal que indicamos en el momento de la creación del proyecto. Esta clase estará definida en el paquete que indicamos durante el proceso de creación.

- *Carpeta res*

Contiene los recursos necesarios para el proyecto (imágenes, vídeos, textos, etc.). Estos recursos se distribuyen en subcarpetas siguiendo la siguiente estructura:

  - *drawable*: Contiene las imágenes, en distintas resoluciones correspondientes a las densidades de las pantallas: *drawable-ldpi* para densidad baja, *drawable-mdpi* para densidad media, *drawable-hdpi* para densidad alta y *drawable-xhdpi* para densidad muy alta.

- *layout*: Contiene los ficheros *xml* que definen las distintas pantallas de la interfaz gráfica. Se distinguen dos tipos: *layout* para una orientación vertical del dispositivo y *layout-land* para una orientación horizontal.
- *anim* y *animator*: Contienen las animaciones.
- *color*: Contiene ficheros *xml* que definen colores según el estado.
- *menú*: Contiene ficheros *xml* con la definición de los menús de la aplicación.
- *xml*: Contiene ficheros *xml* de datos que sean necesarios.
- *raw*: Contiene ficheros que generalmente no son *xml* y que no se incluyan en el resto de carpetas de recursos.
- *values*: Contiene ficheros *xml* de varios tipos, como los que definen las cadenas de texto (*strings.xml*), estilos (*styles.xml*), arrays de valores (*arrays.xml*), etc.

No todas estas carpetas son necesarias. Solo deberán aparecer aquellas que se utilicen en el proyecto. Por defecto, al crear un nuevo proyecto, se incluyen los recursos *activity_main.xml* en la carpeta *layout*, *activity_main.xml* en la carpeta *menú* y *strings.xml* y *styles.xml* en la carpeta *values*.

Se puede observar que algunas carpetas puede crearse varias veces, utilizando algún tipo de sufijo, como por ejemplo la carpeta *values-v11*. Este tipo de carpetas indican que sus recursos se aplicarán sólo a los dispositivos que tengan una versión de Android igual o superior a la indicada. En este caso, el 11 indica la versión 3.0 (API 11).

- *Carpeta gen*

En esta carpeta se incluirán los elementos de código generados automáticamente al compilar el proyecto. Dado que estos ficheros se generan automáticamente cada vez que se compila el proyecto, es importante no modificarlos manualmente.

Entre estos ficheros automáticos, tenemos que destacar la clase *R.java*. Esta clase contendrá en todo momento todos los identificadores de todos los recursos que hayamos definido en la carpeta *res*, definidos en forma de constantes, para poder acceder fácilmente a ellos en cualquier momento. Por ejemplo, la constante *"R.string.app_name"* contendrá identificador del recurso ubicado en *strings.xml* y con el identificador *app_name*.

- *Carpeta assets*

Contendrá ficheros auxiliares necesarios para la aplicación, como por ejemplo ficheros de configuración. La diferencia con la carpeta *res/raw* es que en este caso no se generará ninguna constante en la clase R, mientras que en los recursos de *res/raw* si.

- *Carpeta libs*

Contendrá todas las librerías externas que se requieran para compilar el proyecto. Estas librerías estarán normalmente en formato *jar*.

- *Otros ficheros a destacar*

Finalmente debemos destacar la existencia del fichero *AndroidManifest.xml* definido en el directorio raíz del proyecto. Este fichero contendrá la definición de los diferentes elementos que componen la aplicación, como su nombre, versión, icono, permisos necesarios, etc. El otro fichero que debemos mencionar se trata del ejecutable de la aplicación, que se instalará en los dispositivos. Este fichero llevará por nombre el nombre del proyecto y tendrá como extensión *apk*. Cuando se generé la aplicación, este ejecutable se ubicará en la carpeta *bin* del proyecto.

➢ Componentes de una aplicación

Tras ver cómo se organiza un proyecto Android, tenemos que saber cuáles son los elementos principales sobre los que trabajaremos para definir nuestra aplicación. Estos engloban tanto a aquellos que se encargarán de definir el comportamiento de la aplicación, como a otros que definirán los recursos gráficos. En Android se va a disponer de unos elementos similares a cualquier otro lenguaje de programación (ventanas, servicios, eventos, etc.) pero un poco particulares.

- *Activity*

Se trata del elemento principal de Android. Cada pantalla que mostremos al usuario, vendrá definida por una *actividad* única. Se podría decir que una *actividad* es una ventana o pantalla de cualquier otro lenguaje de programación.

Las *actividades* son clases Java que extienden de la clase *Activity*. En ellas se definen dos métodos principales: *onCreate* y *onCreateOptionsMenu*. El primero de estos métodos se ejecutará en el momento de ejecutar la actividad, es decir, cuando Android cargue la pantalla asociada a la actividad para mostrarla al usuario. El segundo generará el menú que estará disponible si el usuario pulsa el botón *menú* de su dispositivo.

```
public class EjemploActivity extends Activity {

  protected void onCreate(Bundle savedInstanceState) {
    // Código del método
  }

  public boolean onCreateOptionsMenu(Menu menu) {
    // Código del método
  }

}
```

FIGURA 21: EJEMPLO DE UNA ACTIVITY

- *View*

Las *vistas* son los elementos básicos que generan la interfaz gráfica de la aplicación. Estas *vistas* son ficheros *xml*, que guardan cierta similitud con un fichero *html*. En ellas podremos utilizar una gran cantidad de etiquetas que nos permitirán introducir botones, textos, cuadros de texto, imágenes, etc. Además de utilizar estos elementos básicos que proporcionar Android, se pueden crear elementos propios.

El principal elemento de las *vistas* son los *layout*. Estos nos permitirán dividir la pantalla en distintas secciones, en las cuales iremos ubicando los elementos que queramos. Las *vistas* siempre deben comenzar con un elemento *LinearLayout*, que definirá que estructura global tendrá la pantalla.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/ejemploVista" style="@style/FondoPrincipal" android:orientation="vertical" >

   <TextView style="@style/TextoPrincipal" android:text="@string/textoVist" />

   <ImageView style="@style/ImagenPrincipal"  android:src="@drawable/imagenVista" android:contentDescription="@string/textoImagen" />

   <LinearLayout android:layout_width="match_parent" android:layout_height="wrap_content"  android:orientation="horizontal" >

      <Button style="@style/BotonRepetir" android:background="@drawable/botonVista" />

      <TextView style="@style/TextoRepetir" android:text="@string/textoBoton" />
   </LinearLayout>

</LinearLayout>
```

FIGURA 22: EJEMPLO DE UNA VIEW

En la imagen superior podemos ver un pequeño ejemplo. Tenemos el *LinearLayout* inicial, que está definido con una orientación vertical. Esto quiere decir que los elementos que haya dentro del *layout* se mostrarán apilados, cada uno debajo del elemento anterior. Por lo tanto, tendremos un primer elemento que es un texto, que tendrá debajo un elemento que es una imagen, y debajo de la imagen, tenemos otro *layout*, definido con orientación horizontal. Esto hará que los elementos contenidos en

él se muestren uno a continuación del otro, distribuidos en columnas. Por lo tanto, debajo de la imagen tendremos dos columnas, la primera de las cuales contendrá un botón, y la segunda un texto.

La lista de atributos que puede tener cada elemento es muy larga, pero debemos destacar algunos básicos. El atributo *android:id* nos permite asignar un identificador concreto al elemento. Estos identificadores generarán una nueva entrada en la clase *R* comentada anteriormente. Este atributo es opcional, pero resulta importante para poder acceder al elemento en cualquier momento. En cambio, hay otros atributos que resultan obligatorios para un elemento concreto. El atributo *android:text* es obligatorio para los elementos *TextView*, y contendrá el texto a mostrar en el elemento. El atributo *android:src* es obligatorio en el elemento *ImageView*, e indicará la ruta de la imagen que se mostrará. Cualquier atributo puede definirse externamente para ser utilizado como un estilo. Esto resulta útil si vamos a disponer de varios elementos que tengan atributos idénticos. Para hacer referencia a estos estilos, se utiliza el atributo *style*. Para ver una descripción completa de los atributos, se puede visitar la documentación oficial de Android.

- *Service*

Los *servicios* son componentes que no están asociados a ninguna interfaz gráfica. Estos *servicios* se ejecutan en segundo plano para realizar cualquier acción que no requiera una presentación inmediata al usuario, como por ejemplo actualizar datos, mostrar notificaciones cuando lleguen, etc.

- *Intent*

Los *intent* son los elementos básicos de comunicación entre los distintos componentes Android. Se podría decir que son mensajes que son enviados de un componente a otro de la aplicación, o entre distintas aplicaciones. Con un *intent* se puede pasar de una actividad a otra, iniciar un servicio, enviar un mensaje, iniciar otra aplicación, etc.

- *Otros componentes*

Estos cuatro componentes mencionados forman el núcleo de Android. Con ellos se pueden llevar a cabo aplicaciones sencillas. Sin embargo, cuando queremos conseguir

una aplicación más potente, puede ser que necesitemos algo más, por lo que también conviene mencionar otros tres componentes importantes.

El primero de ellos es el *Content Provider* o proveedor de contenidos. Este elemento permite compartir datos entre aplicaciones. El segundo es el *Broadcast Receiver*. Este componente está destinado a detectar y reaccionar ante determinados mensajes o eventos globales, como "Batería baja", "SMS recibido", o mensajes mandados por otras aplicaciones, mediante *intents*. El último de los componentes que merece la pena mencionar corresponde a los *Widgets*. Estos componentes visuales, normalmente interactivos, permiten mostrar información en la pantalla principal del dispositivo y recibir actualizaciones periódicas.

- Desarrollo de una aplicación sencilla

Para observar de forma práctica todo lo mencionado acerca de desarrollar una aplicación en Android, vamos a mostrar cómo hacer un "Hola Usuario". Esta pequeña aplicación constará de dos pantallas, que seguirán un esquema como el siguiente:

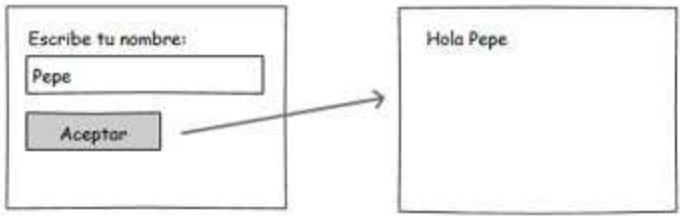


FIGURA 23: DISEÑO DE LA APLICACIÓN DE EJEMPLO

Vamos a asumir que tenemos el proyecto ya creado, siguiendo los pasos mencionados en el apartado correspondiente. En ellos habremos introducido como nombre del proyecto "HolaUsuario" y como nombre de la actividad principal "IntroduceNombre".

Lo primero que hay que hacer, es definir la primera pantalla, que será la encargada de pedir el nombre al usuario. Para ello, buscamos la actividad principal que Eclipse habrá ya creado (recordemos que los ficheros que contienen las interfaces gráficas se encuentran en el directorio */res/layout/*). Al abrir el fichero, veremos que contiene el *layout* principal y un elemento de texto. Reutilizaremos lo que ya hay e iremos incluyendo lo que necesitemos.

En primer lugar, el *layout* principal viene por defecto definido como un *RelativeLayout* y sin orientación, por lo que nosotros lo cambiaremos por *LinearLayout* y le añadiremos el atributo *android:orientation* fijando su valor en vertical, ya que queremos que los tres elementos que compondrán nuestra pantalla inicial se muestren apilados.

Tras esto, vamos a reutilizar el elemento de texto que tenemos, para mostrar el texto "Escribe tu nombre". Para ello, basta con asignar al atributo *android:text* el texto que queremos que muestre. Para hacer esto hay dos formas. Podemos ponerlo directamente (de forma que quede *android:text="Escribe tu nombre"*) o podemos definir el texto en el recurso correspondiente a las cadenas y referenciarlo (de esta forma quedaría algo como *android:text="@strings/escribe_nombre"*). Se recomienda utilizar la segunda forma, ya que de esta manera tendremos todos los textos ubicados en el mismo lugar, por lo que un cambio de texto será fácil de llevar a cabo (por ejemplo, para traducir la aplicación a otro idioma). Además, esto nos permitirá reutilizar el mismo texto en distintas interfaces.

Como aconsejamos utilizar el texto como recurso, vayamos a ver cómo hacer esto. Como vimos en un apartado previo, la definición de las cadenas de texto se lleva a cabo en el fichero *strings.xml* ubicado en */res/values/*. Abrimos ese fichero y añadimos una nueva línea que contenga el texto que nos interesa. Al hacerlo, este fichero quedará similar a lo siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">HolaUsuario</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>

    <string name="escribe_nombre">Escribe tu nombre:</string>

</resources>
```

FIGURA 24: FICHERO STRINGS.XML DE LA APLICACIÓN DE EJEMPLO

Por lo tanto, para hacer referencia a esta cadena en nuestra interfaz, utilizaremos el atributo *android:text="@strings/escribe_nombre"*. La @ indica siempre que se está haciendo referencia a un recurso y el nombre que la acompaña indica cual. Finalmente, después de la / viene el nombre que identifica el elemento dentro del recurso. Una vez hecho esto, debemos eliminar los atributos *android:layout_centerHorizontal* y

android:layout_centerVertical, ya que estos atributos tienen sentido con el *RelativeLayout* inicial, pero no con el *LinearLayout* que pusimos nosotros.

Con esto ya hemos reutilizado los dos elementos iniciales del *layout*, pero debemos añadir más. Concretamente, debemos añadir un cuadro de texto para que el usuario introduzca su nombre y un botón para aceptar el nombre y pasar a la siguiente pantalla. En el primer caso, incluiremos un elemento *EditText* y en el segundo un elemento *Button*, quedando nuestra pantalla inicial similar a la siguiente:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
   android:layout_height="match_parent" tools:context=".IntroduceNombre" android:orientation="vertical" >

   <TextView android:layout_width="wrap_content" android:layout_height="wrap_content"
      android:text="@string/escribe_nombre" />

   <EditText android:id="@+id/textoNombre" android:layout_width="match_parent"
      android:layout_height="wrap_content" android:inputType="text" />

   <Button android:id="@+id/botonHola" android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="@string/aceptar" />

</LinearLayout>
```

FIGURA 25: VIEW INICIAL DE LA APLICACIÓN DE EJEMPLO

En la imagen vamos a destacar varias cosas que son importantes. En los elementos *EditText* y *Button* hemos incluido atributos *android:id*. Esto lo hemos hecho para poder tener una referencia a estos elementos en la clase *R*, para poder referenciarlos posteriormente en cualquier lugar de la aplicación. La realidad es que conviene que todos los elementos contengan su identificador ya que nunca sabes en qué momento futuro puedes necesitar referenciarlos. No obstante, nosotros solo vamos a necesitar referenciar esos dos, así que con eso será suficiente.

Una vez tenemos la primera pantalla, vamos a pasar a definir la segunda. Esto se hará mediante el menú *File* → *New* → *Android XML File*. En la pantalla que nos salga introduciremos el nombre del nuevo fichero (por ejemplo *activity_saludo.xml*) y seleccionaremos como elemento raíz *LinearLayout*. Una vez creado, lo abrimos y añadimos un único elemento de texto que será quién muestre el mensaje de saludo. En este caso añadiremos un identificador ya que tendremos que referenciarlo en el código de la aplicación para añadirle el texto en función del nombre. Debido a que el texto se definirá más tarde, añadiremos el atributo *android:text* pero dejando su valor como una cadena vacía. Al final quedará algo como lo que puede verse a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:layout_width="match_parent"
    android:layout_height="match_parent" android:orientation="vertical" >

  <TextView android:id="@+id/textoSaludo" android:layout_width="wrap_content"
    android:layout_height="wrap_content"  android:text="" />

</LinearLayout>
```

FIGURA 26: VIEW DE RESPUESTA DE LA APLICACIÓN DE EJEMPLO

Una vez tenemos las dos interfaces que definirán nuestra aplicación, debemos definir la lógica que nos indicará como pasar de una a la otra. Para ello se utilizarán, como se comentó anteriormente, las *actividades*. Empezaremos por la *actividad* principal creada por defecto por Eclipse, con el nombre *IntroduceNombre.java* (recordemos que los ficheros con el código fuente se encuentran en la carpeta */src/*).

Al abrirla, encontramos los dos métodos obligatorios ya creados, así que comentaremos un poco lo que hay en cada uno de ellos. En el método *onCreate* vemos que se llama a un constructor de la clase padre, lo cual podemos ignorar porque es la forma de instanciar *actividades* en Android, y posteriormente vemos una línea con una llamada al método *setContentView*. Esta llamada lo que está haciendo es fijar la *vista* que se va a asociar con esta *actividad*, para lo que utiliza el identificador correspondiente definido en la clase *R*. El método *onCreateMenu* no lo utilizaremos. Basta indicar que simplemente se está indicando que menú ha de cargar la aplicación en la *vista* asociada a la *actividad*. Este menú estará definido en la sección de recursos correspondiente (es decir, */res/menú/*).

Como las *actividades* están relacionadas con las *vistas* y nosotros tenemos dos *vistas*, tendremos que crear una segunda actividad. Para hacer esto, seleccionamos el paquete donde se encuentra nuestro código (por ejemplo */src/com.example.holausuario*) y pulsamos sobre *File* → *New* → *Class*. En la pantalla que se nos abre, introduciremos el nombre de la nueva *actividad* (por ejemplo *SaludoActivity*) y en el cuadro “superclass” pondremos *android.app.Activity*. Esto quiere decir que nuestra nueva clase extenderá de la clase *Activity* de Android. Cuando hayamos creado la clase, introduciremos el mismo código que hay en la *actividad* principal, con los dos métodos, solo que en este caso, la llamada al método *setContentView*, en lugar de llevar el identificador de la *vista* de la primera pantalla, llevará el de la segunda. Al final deberá quedar algo como:

```
package com.example.holausuario;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;


public class SaludoActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_saludo);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.activity_introduce_nombre, menu);
    return true;
  }

}
```

FIGURA 27: ACTIVITY INICIAL DE LA APLICACIÓN DE EJEMPLO

Con esto ya tenemos el código que nos permite generar las distintas *vistas* de la aplicación. Ahora tendremos que comenzar a incluir la lógica para que se pueda pasar de una a la otra. Para ello, volvemos a la *actividad* inicial y comenzamos a añadir lo que necesitemos.

La aplicación deberá reaccionar cuando el usuario pulse el botón “Aceptar” y deberá recoger el nombre que se ha introducido en el cuadro de texto. Para poder trabajar con esos dos elementos, botón y cuadro de texto, lo primero que debemos hace es obtener una referencia de ellos. Esto se podrá hacer gracias a los identificadores que añadimos en el momento de crear la *vista*. En el código, dentro del método *onCreate* añadiremos dos variables a las que asignaremos las referencias a esos elementos. Concretamente, estas variables serán de tipo *Button* y *EditText*. Para asignar las referencias hay que utilizar el método *findViewById*, al cual se le pasará como parámetro el identificador del elemento definido en la clase *R*. Tras hacer esto, el método *onCreate* quedará similar a:

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_introduce_nombre);

  //Obtenemos una referencia a los controles de la interfaz
  final EditText textoNombre = (EditText)findViewById(R.id.textoNombre);
  final Button botonHola = (Button)findViewById(R.id.botonHola);
}
```

FIGURA 28: MÉTODO ONCREATE DE LA ACTIVITY INICIAL (I)

Con las referencias ya disponibles en nuestras variables, únicamente tenemos que darles el comportamiento que queramos. En nuestro caso, hay que reaccionar cuando se

pulse el botón. Estas reacciones se implementan mediante objetos denominados *Listener*. Un *Listener* es un objeto que se queda esperando, a la "escucha", hasta que se produce el evento que espera. En nuestro caso, que queremos esperar hasta que se pulse el botón, el *Listener* apropiado es *OnClickListener*, que se fijará a la variable correspondiente al botón mediante el método *setOnClickListener*, quedando el método *onCreate*:

```
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_introduce_nombre);

 // Obtenemos una referencia a los controles de la interfaz
 final EditText textoNombre = (EditText) findViewById(R.id.textoNombre);
 final Button botonHola = (Button) findViewById(R.id.botonHola);

 // Implementamos el evento "click" del botón
 botonHola.setOnClickListener(new OnClickListener() {
   // Código del Listener
 });
}
```

FIGURA 29: MÉTODO ONCREATE DE LA ACTIVITY INICIAL (II)

Una vez asignado el *Listener*, tendremos que decirle que tendrá que hacer cuando se pulse el botón. En nuestro caso habrá que recoger el nombre que se haya introducido en el cuadro de texto, y pasárselo a la siguiente *actividad*, que lo mostrará en su vista.

Como comentamos anteriormente, la comunicación entre *actividades* se realiza mediante los *Intent*, por lo que tendremos que crearnos uno que se encargue de comunicar el nombre introducido. El constructor del *Intent* requiere dos parámetros, una referencia a la *actividad* que está creándolo, y la clase de la *actividad* destino del mensaje. Cuando hayamos creado el *Intent*, debemos obtener el nombre, para lo que utilizaremos la variable creada anteriormente con la referencia del cuadro de texto. Tras obtenerlo, tendremos que añadirlo al *Intent* para que viaje con el mensaje hasta la otra actividad. Esto se realiza mediante un objeto *Bundle*. Estos objetos almacenan pares de clave-valor, por lo que nosotros almacenaremos un par con clave "nombre" y como valor el nombre obtenido. Finalmente nos queda añadir el *Bundle* al *Intent* y arrancar la siguiente actividad mediante una llamada a *startActivity*, pasando como parámetro el *Intent*. Tras todo esto, el método *onCreate* debería verse similar a:

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_introduce_nombre);

  // Obtenemos una referencia a los controles de la interfaz
  final EditText textoNombre = (EditText) findViewById(R.id.textoNombre);
  final Button botonHola = (Button) findViewById(R.id.botonHola);

  // Implementamos el evento "click" del botón
  botonHola.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
      // Creamos el Intent
      Intent intent = new Intent(IntroduceNombre.this, SaludoActivity.class);

      // Creamos la información a pasar entre actividades
      String nombre = textoNombre.getText().toString();
      Bundle b = new Bundle();
      b.putString("nombre", nombre);

      // Añadimos la información al intent
      intent.putExtras(b);

      // Iniciamos la nueva actividad
      startActivity(intent);
    }
  });
}
```

FIGURA 30: MÉTODO ONCREATE DE LA ACTIVIDAD INICIAL (III)

Una vez finalizada la lógica de la primera *actividad*, pasamos a la lógica de la segunda. Esta vez, lo que queremos es recoger la información que nos llega en el *Intent* para extraer el nombre, y mostrar ese nombre en el elemento de texto definido en la segunda *vista*. Lo primero, al igual que en la primera *actividad*, es crear las variables que tomen la referencia de los elementos de la *vista* que nos interesen. En este caso únicamente tendremos una correspondiente al elemento de texto. A continuación, extraeremos la información contenida en el *Intent* que vendrá en un objeto *Bundle*. Finalmente sólo nos quedará fijar el texto de la variable correspondiente al elemento de texto de la *vista*. Tras todo ello, el método *onCreate* de la segunda *actividad* debería quedar algo similar a:

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_saludo);

  // Localizar los controles
  TextView textoSaludo = (TextView) findViewById(R.id.textoSaludo);

  // Recuperamos la información pasada en el intent
  Bundle bundle = this.getIntent().getExtras();

  // Construimos el mensaje a mostrar
  textoSaludo.setText("Hola " + bundle.getString("nombre"));
}
```

FIGURA 31: MÉTODO ONCREATE DE LA ACTIVITY DE RESPUESTA

Ya tenemos toda la lógica de nuestra aplicación creada. Para que funcione la aplicación solo nos queda una cosa por hacer. Como comentamos en una sección previa, existe un archivo especial llamado *AndroidManifest.xml*, el cual contiene los diferentes elementos que componen la aplicación. Algunos de estos elementos son las *actividades*. Por lo tanto tendremos que añadir su definición en este fichero. Para ello se utiliza la etiqueta <*activity*>, dentro de la cual se definirá el nombre de la clase que implementa la actividad (atributo *android:name*) y un título (atributo *android:label*). La primera *actividad* se creó por defecto, por lo que simplemente tendremos que añadir la segunda. El título lo añadiremos al recurso *strings.xml* como ya hicimos anteriormente, por lo que el fichero *AndroidManifest.xml* con esta actividad añadida será similar a:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.holausuario"
    android:versionCode="1" android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="17" />

    <application
        android:allowBackup="true" android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.holausuario.IntroduceNombre" android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".SaludoActivity" android:label="@string/app_name"/>
    </application>

</manifest>
```

FIGURA 32: FICHERO ANDROIDMANIFEST.XML DE LA APLICACIÓN DE EJEMPLO

Para probar nuestra aplicación, vamos a utilizar el simulador de dispositivos AVD. Para crear un nuevo AVD basta con seguir las indicaciones mostradas en los apartados previos. Una vez creado, iremos al menú *Run* → *Run configurations* y en la ventana que nos sale, haciendo *click* con el botón derecho sobre *Android Application* seleccionamos *New*. Introduciremos un nombre y seleccionaremos el proyecto que queremos ejecutar. En la pestaña *Target*, nos aparecerán los AVD que tengamos definidos. En esta pestaña tenemos dos opciones: podemos seleccionar la opción *Always prompt to pick device*, haciendo que cada vez que se ejecute la aplicación tengamos que seleccionar el AVD, o podemos seleccionar la opción *Automatically pick compatible device*, que hará que siempre se ejecute con el mismo dispositivo, el cual seleccionaremos de entre los

disponibles. Si se quiere probar en varios dispositivos distintos, conviene optar por la primera opción, así se podrá cambiar en cada ejecución de dispositivo. Cuando se haya configurado el modo de ejecución, basta con pulsar el botón *Run* para lanzar la aplicación. Si se ha optado por la opción de seleccionar el AVD en cada ejecución, aparecerá una pantalla para escoger el dispositivo de simulación. Una vez arranque el simulador, aparecerá nuestra aplicación en primer plano, y podremos utilizarla como cualquier usuario de un dispositivo real.

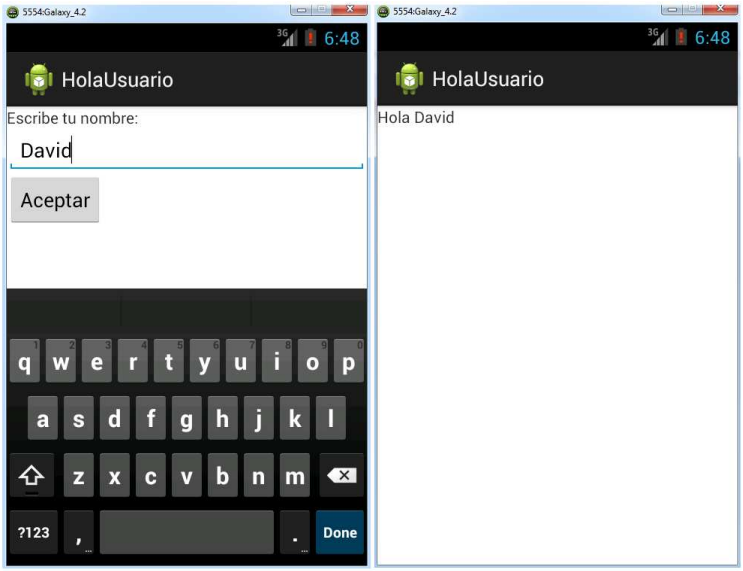


FIGURA 33: RESULTADO FINAL DE LA APLICACIÓN DE EJEMPLO

2.4. ANÁLISIS DE APLICACIONES SIMILARES

A la hora de enfrentarse al diseño de una aplicación móvil, no es suficiente con buscar una aplicación que ofrezca a los usuarios una serie de opciones útiles en algún momento de necesidad, también hay que tener en cuenta la competencia que tienes, ya que el mercado de aplicaciones es muy grande. Para enfrentar esta problemática, se ha realizado un pequeño estudio de las aplicaciones que ya se encuentran disponibles para la plataforma Android, y que pueden competir directamente con la aplicación propuesta.

En primer lugar, se ha optado por realizar una búsqueda en el mercado de aplicaciones oficial de Android de aquellas relacionadas con accidentes de tráfico. Dentro de esta búsqueda se encuentran principalmente las aplicaciones de las distintas compañías de seguros. Entre ellas podemos destacar:

- En Ruta: Se trata de la aplicación de la aseguradora Linea Directa. Ofrece una variedad de servicios divididos en dos grupos, los correspondientes a clientes

de la compañía y los disponibles para todos los usuarios. Entre este último grupo tenemos la posibilidad de acceder a una guía de cómo actuar en caso de accidente, información sobre como rellenar un parte de accidente, una guía de teléfonos útiles y un servicio para localizar el lugar dónde se aparco el vehículo así como de aviso en caso de que se termine el tiempo de estacionamiento pagado.

  - Puntos fuertes: Posibilidad de localizar talleres, gasolineras y centros médicos así como dos utilidades para ayudarnos una vez aparcamos el coche. La primera de ellas para localizar dónde se ha aparcado el vehículo y la segunda para avisarnos cuando haya que renovar el ticket del aparcamiento.
  - Puntos débiles: Muestra puntos débiles en cuanto a la guía en caso de accidente. Esta se centra en como tomar datos del accidente principalmente.

- Mapfre: Esta aplicación, correspondiente a la compañía del mismo nombre, ofrece multitud de servicios, similares a la aplicación anterior. Con ella podremos buscar talleres, oficinas, clínicas, etc., solicitar asistencia en carretera, capturar partes, guardar documentos relacionados con el seguro y otros servicios propios de la compañía.
  - Puntos fuertes: Al igual que con la anterior aplicación, esta también ofrece la posibilidad de encontrar los centros médicos más cercanos y el sistema de localización del lugar dónde se ha aparcado el vehículo y de aviso para renovar el ticket. Además proporciona una serie de consejos útiles sobre conducción.
  - Puntos débiles: Las secciones de asistencia se limitan a indicar que deben hacer en caso de accidente en cuanto al ámbito de la compañía de seguros y fuera de eso, solo permiten enviar la localización para ponerse posteriormente en contacto con el cliente.
- Axa: Esta aplicación de la compañía Axa, resulta una versión similar a las anteriores, pero más cerrada, ya que muchos de sus servicios son accesibles únicamente si proporcionas datos de cliente.
  - Puntos fuertes: Al igual que las anteriores aplicaciones, proporciona la posibilidad de encontrar hospitales y farmacias y también proporciona un

botón directo en la pantalla de inicio para ponerse en contacto con los servicios de emergencias.

- Puntos débiles: Las secciones de siniestros y asistencia solo son accesibles si dispones de un producto de esta compañía, lo que da que pensar (no se ha podido entrar en esos servicios) que están orientados únicamente a trámites con la aseguradora.

Podría decirse a modo de resumen de estas aplicaciones, que todas ellas están orientadas hacia los productos que ofertan a sus clientes, sin preocuparse de si resultan útiles a usuarios ajenos a las compañías. Viendo además que este tipo de aplicaciones no mostraba una gran asistencia en cuanto a la forma de actuar tras un accidente, se ha optado por buscar también aquellas aplicaciones relacionadas con los primeros auxilios, de nuevo en el mercado de aplicaciones oficial de Android. De entre las encontradas, se pueden destacar las siguientes:

- Primeros auxilios (bronczesko): Se trata de una aplicación muy sencilla, que permite movernos por distintas pantallas que muestran como realizar una RPC.
  - Puntos fuertes: Muestra una descripción muy completa de cómo realizar una reanimación cardio-pulmonar.
  - Puntos débiles: No dice nada más sobre otro tipo de necesidades de auxilio y sólo muestra las instrucciones en inglés.
- First Aid (SMART-E): Aplicación muy completa sobre gran variedad de situaciones médicas que pueden afectar a los usuarios.
  - Puntos fuertes: Muestra una gran variedad de categorías de las cuales podemos obtener como actuar frente a ellas.
  - Puntos débiles: Parece estar diseñada para resoluciones concretas, de forma que en un dispositivo móvil de pantalla reducida no se termina de ver todo correctamente.

La mayoría de las aplicaciones dentro del ámbito de los primeros auxilios, están orientadas a tener una información médica básica sobre el usuario, de manera que en caso de accidente, los servicios de emergencias sepan si tiene algún tipo de alergia, tipo de sangre, etc.

3

ANÁLISIS DEL SISTEMA

3. ANÁLISIS DEL SISTEMA

3.1. DESCRIPCIÓN DEL SISTEMA

El sistema a diseñar queda enmarcado en el ámbito de la seguridad vial y de los primeros auxilios, por lo que estará orientada principalmente a todo tipo de usuarios que sean conductores de vehículos. Además, tendrá la flexibilidad de poder ser útil a cualquier viandante, ya que informará no solo de cómo actuar en caso de sufrir un accidente, sino también en el caso de ver un accidente. Por lo tanto, como resumen, la aplicación debe cumplir con tres objetivos principales:

- Proporcionar una serie de pasos a seguir para actuar en caso de ver un accidente.
- Proporcionar una serie de pasos a seguir para actuar en caso de sufrir un accidente.
- Permitir el almacenamiento y la recuperación de información relativa a un accidente.

Este conjunto de objetivos va a permitir a la aplicación completar las carencias mostradas por las aplicaciones analizadas previamente y que pueden considerarse su competencia más directa. Permitirá al usuario actuar de forma concreta ante un accidente, incluyendo algunos pasos básicos de primeros auxilios centrados estos en aquellos percances médicos derivados de un accidente. Además, permitirá al usuario almacenar información del accidente, solo que en el caso de nuestra aplicación, será información genérica, sin estar sujeta a las necesidades de una compañía de seguros concreta. Esto dotará al sistema de una gran flexibilidad y no lo atará a un modelo de datos concreto.

Para lograr los dos primeros objetivos nombrados, se utilizarán imágenes y audios descriptivos de los pasos que debe seguir el usuario en cada momento, facilitándole una reacción rápida y una forma sencilla de pasar de un paso al siguiente o incluso volver atrás para volver a visualizar las acciones de pasos previos. Adicionalmente, para poder ilustrar las necesidades más complejas de primeros auxilios, se incluirán vídeos explicativos.

En cuanto al tercer objetivo, correspondiente al almacenamiento de datos, se optará por almacenar aquellos datos importantes que el usuario puede requerir a posteriori para completar la información requerida por su compañía de seguros. Estos datos incluirán,

por ejemplo, la información del lugar y la fecha del accidente. Para poder ubicar el lugar del accidente, será necesario que la aplicación tenga acceso a internet y/o el GPS. En caso de sufrir un accidente con contrario, se permitirá almacenar la información más necesaria de este, como puede ser su nombre y teléfono de contacto, marca, modelo y matrícula del coche y datos del seguro. No será necesario incluir todos los datos pedidos.

Dado que la aplicación debe utilizarse en un ambiente que se puede considerar de tensión, debe de hacerse especial hincapié en que la interfaz sea lo más amigable y sencilla posible, evitando que el usuario tenga dudas sobre cómo utilizarla. Para facilitar esto, se proporcionará una ayuda que pueda ser consultada por los usuarios, indicando en ella las principales características de las pantallas y el uso de cada uno de sus elementos. Además, las distintas pantallas serán sencillas, evitando en todo momento elementos que puedan distraer al usuario de la información de la acción a realizar.

3.2. ANÁLISIS DE REQUISITOS

Vamos a dividir los requisitos del sistema en dos grupos: requisitos funcionales y requisitos no funcionales. Los primeros engloban los requisitos que especifican que debe hacer el sistema, mientras que los segundos se refieren a características del sistema que han de cumplirse para que sea válido.

El formato de las tablas de requisitos será el siguiente:

Código: Nombre	
Prioridad	
Descripción	

TABLA 2: FORMATO TABLAS REQUISITOS

- Código: Incluirá la cadena RF o RNF, para requisitos funcionales y no funcionales respectivamente, seguida de un número. Este número comenzará en el 1 e irá incrementándose en los sucesivos requisitos.
- Nombre: Nombre que dé una idea clara del objetivo del requisito.
- Prioridad: Se trata de la importancia que tiene el requisito para el correcto funcionamiento del sistema. Los requisitos se dividirán en tres categorías, según su importancia: Esencial, Deseable y Opcional.
- Descripción: Incluirá una descripción completa del requisito.

- Requisitos funcionales

RF-01: Textos informativos	
Prioridad	Esencial
Descripción	Los pasos informativos deberán introducir un texto descriptivo corto que indique al usuario que debe hacer.

TABLA 3: RF-01: TEXTOS INFORMATIVOS

RF-02: Imágenes	
Prioridad	Esencial
Descripción	Los pasos informativos deberán contar con imágenes descriptivas de la acción que debe llevar a cabo el usuario.

TABLA 4: RF-02: IMÁGENES

RF-03: Audios	
Prioridad	Esencial
Descripción	Los pasos informativos deberán contar con un audio que enuncie la acción que debe llevar a cabo el usuario.

TABLA 5: RF-03: AUDIOS

RF-04: Repetir audio	
Prioridad	Deseable
Descripción	Los pasos informativos podrán incluir un botón que permita que se vuelva a reproducir el audio de ese paso.

TABLA 6: RF-04: REPETIR AUDIO

RF-05: Finalizar audio	
Prioridad	Esencial
Descripción	Los audios que se estén reproduciendo en un determinado paso deberán finalizar su reproducción inmediatamente en caso de que el usuario cambie de paso.

TABLA 7: RF-05: FINALIZAR AUDIO

RF-06: Vídeos	
Prioridad	Deseable
Descripción	Algunos pasos podrán contar con vídeos en lugar de imágenes, si se considera que estos son necesarios para indicar con claridad la acción que debe llevar a cabo el usuario.

TABLA 8: RF-06: VÍDEOS

RF-07: Controles para vídeos	
Prioridad	Deseable
Descripción	En caso de incluir vídeos, estos deberán contar con los controles básicos para la reproducción y pausa.

TABLA 9: RF-07: CONTROLES PARA VÍDEOS

RF-08: Finalizar vídeo	
Prioridad	Deseable
Descripción	En caso de incluir vídeos, estos deberán finalizar su reproducción inmediatamente en caso de que el usuario cambie de paso.

TABLA 10: RF-08: FINALIZAR VÍDEO

RF-09: Preguntas sencillas	
Prioridad	Esencial
Descripción	Los pasos interrogativos deberán proporcionar preguntas sencillas cuyas respuestas sean “Si” o “No” para dirigir los pasos en un camino u otro.

TABLA 11: RF-09: PREGUNTAS SENCILLAS

RF-10: Mostrar mapa	
Prioridad	Esencial
Descripción	Los pasos que siga el usuario deben siempre llegar, antes o después, a un punto en el cual se muestra la ubicación del accidente mediante un mapa.

TABLA 12: RF-10: MOSTRAR MAPA

RF-11: Mostrar ubicación	
Prioridad	Opcional
Descripción	En el mapa que muestre la posición del accidente, se mostrarán opcionalmente las coordenadas del accidente.

TABLA 13: RF-11: MOSTRAR UBICACIÓN

RF-12: Llamada a emergencias	
Prioridad	Esencial
Descripción	Los pasos que muestren el mapa con la ubicación del accidente deberán incluir además un botón mediante el cual se pueda llamar directamente al teléfono de emergencias.

TABLA 14: RF-12: LLAMADA A EMERGENCIAS

RF-13: Guardar datos del accidente	
Prioridad	Esencial
Descripción	En caso de que se haya tenido un accidente, se deberá permitir al usuario almacenar información relativa al mismo de manera que pueda recuperarse posteriormente. Esta información deberá incluir la fecha y la hora en la que se ha producido el accidente y una descripción del mismo.

TABLA 15: RF-13: GUARDAR DATOS DEL ACCIDENTE

RF-14: Guardar datos del asegurado	
Prioridad	Esencial
Descripción	En caso de que se haya tenido un accidente y este haya sido contra otro vehículo, se deberá permitir al usuario almacenar información relativa a los datos de asegurado del contrario. Esta información deberá incluir un número de teléfono y datos del contrario.

TABLA 16: RF-14: GUARDAR DATOS DEL ASEGURADO

RF-15: Tomar fotos del accidente	
Prioridad	Esencial
Descripción	En caso de que se haya tenido un accidente, se deberá permitir al usuario que haga fotos del mismo para que posteriormente puedan ser visualizadas.

TABLA 17: RF-15: TOMAR FOTOS DEL ACCIDENTE

RF-16: Acceso a datos guardados	
Prioridad	Esencial
Descripción	Se deberá permitir al usuario acceder a los datos guardados previamente, incluidos los datos del accidente, los datos del asegurado y las fotos tomadas.

TABLA 18: RF-16: ACCESO A DATOS GUARDADOS

RF-17: Ayuda	
Prioridad	Esencial
Descripción	Se deberá incluir una sección de ayuda básica que muestre al usuario como utilizar la aplicación.

TABLA 19: RF-17: AYUDA

RF-18: Navegación	
Prioridad	Esencial
Descripción	Se deberá permitir al usuario navegar entre los distintos pasos deslizando el dedo de derecha a izquierda o de izquierda a derecha, en función de si quiere avanzar al siguiente paso o al anterior respectivamente. En el caso de preguntas, solo se deberá permitir retroceder al paso anterior mediante deslizado, mientras que para avanzar deberá pulsar una de las respuestas.

TABLA 20: RF-18: NAVEGACIÓN

- Requisitos no funcionales

Código: RNF-01 Pasos en caso de ver un accidente	
Prioridad	Esencial
Descripción	Se deberán especificar los pasos que debe seguir el usuario para actuar correctamente en caso de ver un accidente.

TABLA 21: RNF-01: PASOS EN CASO DE VER UN ACCIDENTE

RNF-02: Pasos en caso de sufrir un accidente	
Prioridad	Esencial
Descripción	Se deberán especificar los pasos que debe seguir el usuario para actuar correctamente en caso de sufrir un accidente.

TABLA 22: RNF-02: PASOS EN CASO DE SUFRIR UN ACCIDENTE

RNF-03: Pasos informativos	
Prioridad	Esencial
Descripción	Se deberá contar con pasos que informen al usuario de la acción que tiene que llevar a cabo en ese momento.

TABLA 23: RNF-03: PASOS INFORMÁTIVOS

RNF-04: Pasos interrogativos	
Prioridad	Esencial
Descripción	Se deberá contar con pasos interrogativos que permitan al usuario avanzar en unas acciones u otras según la situación que esté viviendo.

TABLA 24: RNF-04: PASOS INTERROGATIVOS

RNF-05: Botón emergencias claro	
Prioridad	Esencial
Descripción	El botón para llamar al teléfono de emergencias deberá ser claramente distinguible.

TABLA 25: RNF-05: BOTÓN EMERGENCIAS CLARO

RNF-06: Versión mínima android	
Prioridad	Esencial
Descripción	Se deberá desarrollar la aplicación de forma que sea compatible desde la versión 2.2 de Android en adelante.

TABLA 26: RNF-06: VERSIÓN MÍNIMA ANDROID

RNF-07: Funcionamiento en móviles	
Prioridad	Esencial
Descripción	La aplicación deberá estar diseñada para ejecutar de manera óptima en dispositivos de pantalla pequeña.

TABLA 27: RNF-07: FUNCIONAMIENTO EN MÓVILES

RNF-08: Funcionamiento en tablets	
Prioridad	Opcional
Descripción	La aplicación podrá estar diseñada para ejecutar de manera óptima en dispositivos de pantallas de grandes dimensiones similares a tablets.

TABLA 28: RNF-08: FUNCIONAMIENTO EN TABLETS

RNF-09: Funcionamiento apaisado	
Prioridad	Opcional
Descripción	La aplicación podrá estar diseñada para ejecutar de manera óptima en caso de que se rote la vista del dispositivo para visualizar la interfaz de manera apaisada.

TABLA 29: RNF-09: FUNCIONAMIENTO APAISADO

3.3. CASOS DE USO

En la aplicación se van a distinguir un total de cuatro casos de uso principales. Estos casos de uso se ven reflejados en el siguiente diagrama:

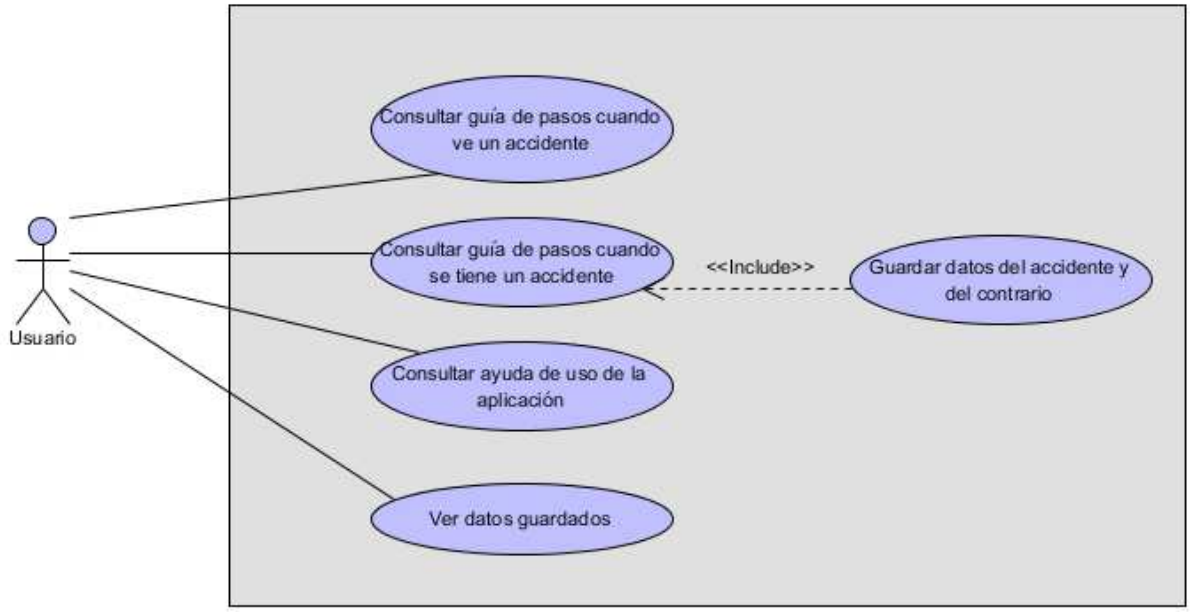


FIGURA 34: CASOS DE USO

- Consultar guía de pasos cuando ve un accidente

En este caso de uso, el usuario podrá observar la serie de acciones que debe llevar a cabo tras haber visto un accidente ajeno. A lo largo de los pasos, el usuario se encontrará con diversas preguntas que permitirán encauzar las acciones en un sentido u otro de forma que realice las acciones más oportunas. Dentro de este caso de uso, las primeras acciones informarán sobre qué hacer para asegurar que la zona del accidente está en buenas condiciones de seguridad, para a continuación llevar al usuario a un paso que incluye un mapa en el cual podrá ver la ubicación dónde se encuentra actualmente, junto con las coordenadas y un botón para llamar a los servicios de emergencias. Finalmente, en caso de que haya heridos en el accidente, este caso de uso también mostrará unos pasos mediante los cuales el usuario podrá ofrecer los primeros servicios de auxilio básicos a los heridos. A continuación podemos ver el diagrama de flujo completo de este caso de uso:

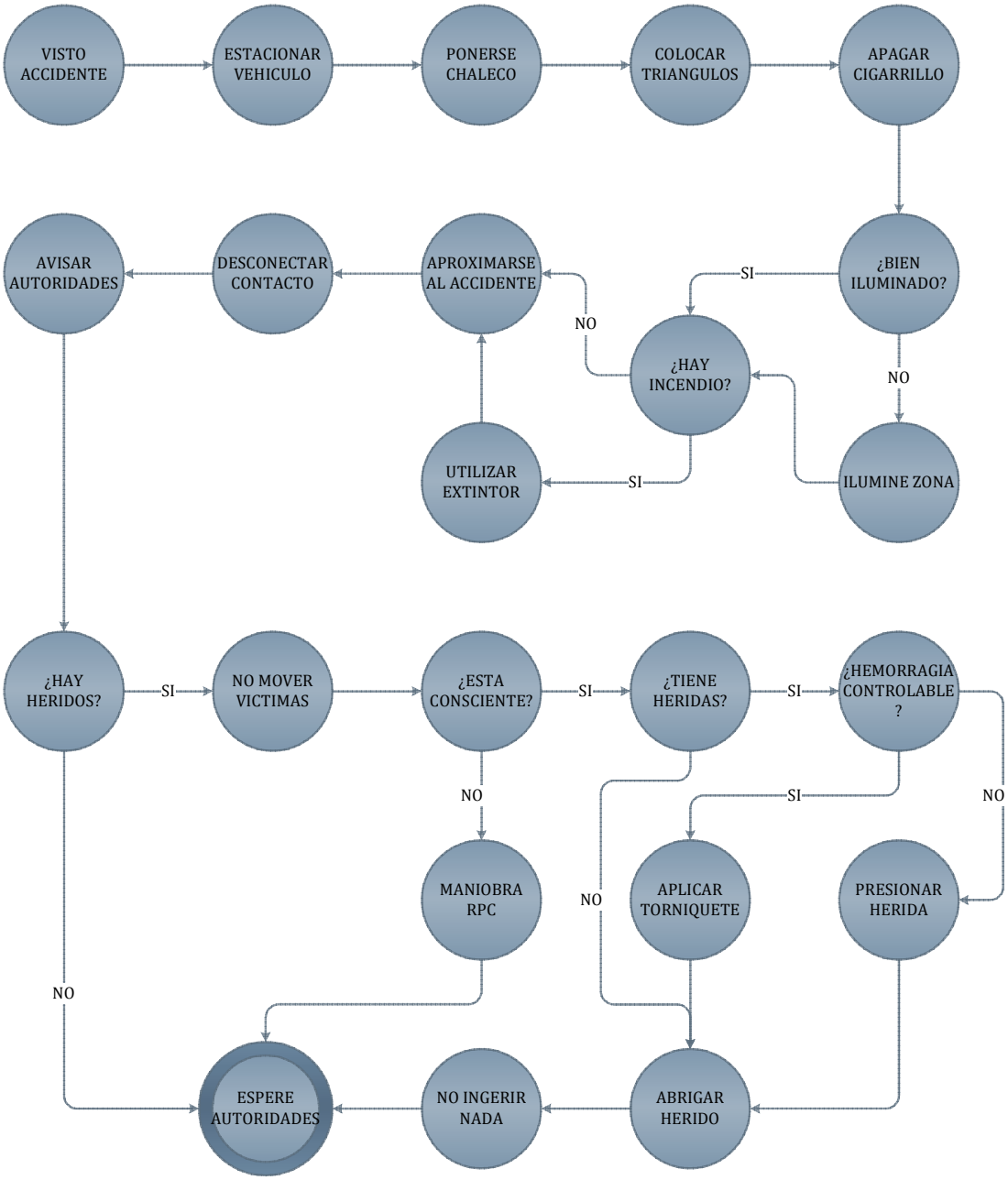


FIGURA 35: FLUJO DE PASOS A SEGUIR EN CASO DE VER UN ACCIDENTE

- Consultar guía de pasos cuando se tiene un accidente

En este caso de uso, se proporcionarán al usuario una serie de pasos a seguir en caso de sufrir un accidente propio. El usuario verá dos opciones bien diferenciadas en este caso de uso. La primera de ellas, corresponde a la situación en la que haya sufrido un accidente y además esté herido. En este caso se le ofrece inmediatamente la posibilidad de llamar a los servicios de emergencias a la vez que se le muestra un mapa con la ubicación en la que se encuentra para que la notifique. La segunda

opción se centra en los casos en los que el usuario haya sufrido un accidente pero no se encuentre herido. En esta situación, se mostrarán los pasos básicos para asegurarse de que no sufre peligro tras el accidente, para posteriormente mostrarle la opción de avisar a los servicios de emergencias junto con el mapa indicando la posición. Además, esta situación incluye una opción para guardar información del accidente, así como los datos de asegurado de un posible contrario con el que haya sufrido el accidente. También cabe la posibilidad de hacer fotos del accidente. A continuación se muestra el diagrama de flujo completo para este caso de uso:

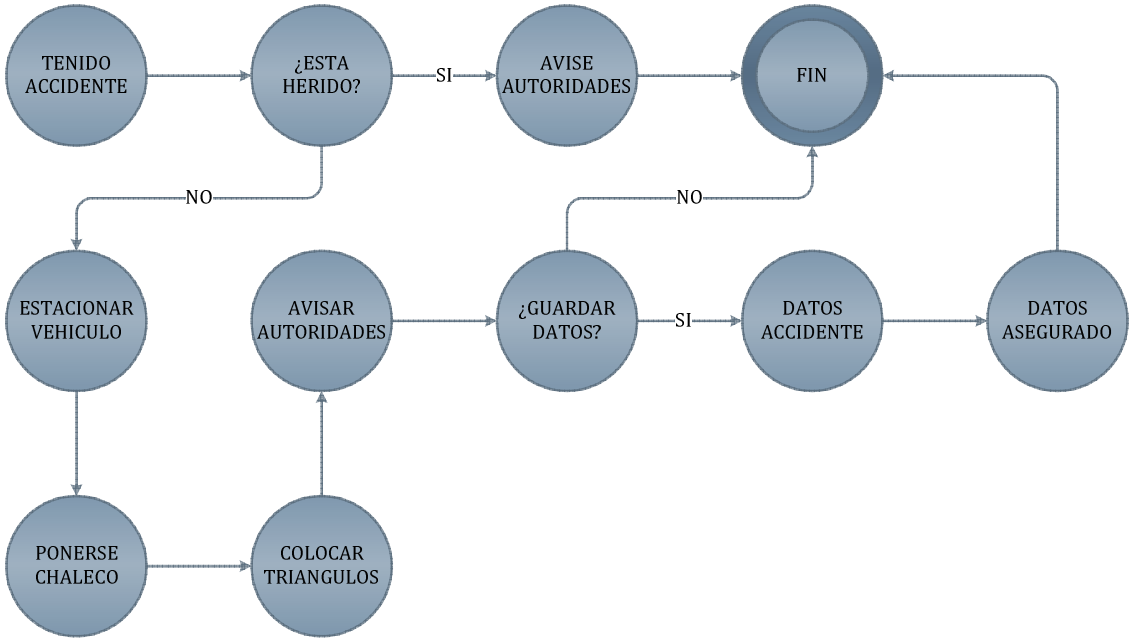


FIGURA 36: FLUJO DE PASOS A SEGUIR EN CASO DE TENER UN ACCIDENTE

- Consultar ayuda de uso de la aplicación

En el caso de uso correspondiente a la consulta de la ayuda de la aplicación, se mostrarán al usuario una serie de pantallas descriptivas de la información que va a recibir en cada tipo de pantalla y como puede interactuar con cada una de ellas según su contenido. Las ayudas se mostrarán de forma sucesiva, hasta mostrar la información relativa al proyecto. El flujo correspondiente a este caso de uso se corresponde con el siguiente esquema:

FIGURA 37: FLUJO DE PASOS A SEGUIR AL CONSULTAR LA AYUDA

- Ver datos guardados

Este caso de uso se basa en haber guardado previamente datos de al menos un accidente. Concretamente, el caso de uso nos proporcionará una lista de los accidentes que tenemos guardados, indicando la fecha del accidente guardada en su momento. El usuario podrá seleccionar uno de los accidentes de forma que visualice los datos del mismo, así como los datos del contrario si es que lo hubo. Además, podrá visualizar las fotos del accidente. A continuación se muestra el flujo de este caso de uso:

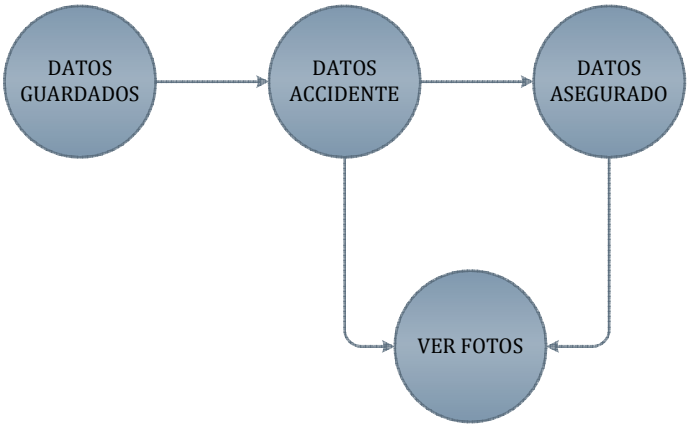


FIGURA 38: FLUJO DE PASOS A SEGUIR AL GUARDAR DATOS DE UN ACCIDENTE

3.4. PLANIFICACIÓN

Para la planificación del desarrollo, vamos a dividir este en distintas iteraciones, según los principios de las metodologías ágiles. Cada una de ellas comprenderá un ciclo de desarrollo de una parte concreta de la aplicación, de forma que al terminar cada una, se tenga una nueva versión del producto con una demo funcional, que permita seguir evaluando los requisitos del sistema. Cada iteración estará dividida en tareas que han de llevarse a cabo. Cada tarea se describirá conforme a la siguiente tabla:

Código	Nombre
Descripción	
Dependencias	
Estimado:	Estado:

TABLA 30: FORMATO TABLA DE TAREA

- Código: Identificará la tarea unívocamente para poder referenciarla posteriormente desde otras tareas. Tendrá el formato Tarea-XX.
- Nombre: Un nombre corto que sirva para entender el propósito de la tarea que se va a llevar a cabo.
- Descripción: Especificará de forma clara el trabajo que se debe llevar a cabo en la tarea.
- Dependencias: Indicará las dependencias con otras tareas que han de estar validadas antes de poder desarrollar la tarea.
- Estimado: Tiempo estimado en horas para desarrollar la tarea.
- Estado: Estado en el que se encuentra la tarea. Los estados pueden ser: TODO (indica que la tarea está por hacerse), IN PROGRESS (indica que la tarea se está desarrollando), DONE (indica que la tarea se ha terminado de desarrollar) y VALIDATED (indica que se ha validado el funcionamiento de la tarea).

Las iteraciones se van a planificar de forma que la suma de las estimaciones de tiempos para todas las tareas que las compongan sea igual a 30 horas.

➢ Iteración 1

Tarea-01	Configurar el entorno de desarrollo
Descripción	Descargar, instalar y configurar el software necesario para el desarrollo de aplicaciones Android, lo que incluye el SDK de Android, plugin para desarrollar en Eclipse, además de la definición de los terminales AVD para realizar las pruebas.
Dependencias	-
Estimado: 6	Estado: TODO

TABLA 31: TAREA-01: CONFIGURAR EL ENTORNO DE DESARROLLO

Tarea-02	**Diseñar las pantallas de preguntas**
Descripción	Realizar el diseño visual de las pantallas que vayan a albergar preguntas para encauzar al usuario en un camino u otro.
Dependencias	Tarea-01
Estimado:	**3** **Estado:** **TODO**

TABLA 32: TAREA-02: DISEÑAR LAS PANTALLAS DE PREGUNTAS

Tarea-03	**Diseñar las pantallas de acciones**
Descripción	Realizar el diseño visual de las pantallas que vayan a albergar la información sobre las acciones que debe realizar el usuario en un momento dado
Dependencias	Tarea-01
Estimado:	**3** **Estado:** **TODO**

TABLA 33: TAREA-03: DISEÑAR LAS PANTALLAS DE ACCIONES

Tarea-04	**Implementar la primera actividad de pregunta**
Descripción	Implementación de la actividad que corresponderá a la primera pantalla que se mostrará de la aplicación, correspondiente a la pregunta “¿Qué tipo de incidencia ha ocurrido?”
Dependencias	Tarea-02
Estimado:	**1** **Estado:** **TODO**

TABLA 34: TAREA-04: IMPLEMENTAR LA PRIMERA ACTIVIDAD DE PREGUNTA

Tarea-05	**Implementar la primera actividad de acción**
Descripción	Implementación de la actividad que corresponderá con la pantalla “Estacionar vehículo” que se mostrará al seleccionar la respuesta “He visto un accidente” a la pregunta “¿Qué tipo de incidencia ha ocurrido?”
Dependencias	Tarea-03
Estimado:	**1** **Estado:** **TODO**

TABLA 35: TAREA-05: IMPLEMENTAR LA PRIMERA ACTIVIDAD DE ACCIÓN

Tarea-06	**Implementar evento click en primera pregunta**
Descripción	Capturar cuando el usuario pulse el botón de respuesta “He visto un accidente” en la pregunta “¿Qué tipo de incidencia ha ocurrido?” y pasar a la actividad “Estacionar vehículo”.
Dependencias	Tarea-04 Tarea-05
Estimado:	**1** **Estado:** **TODO**

TABLA 36: TAREA-06: IMPLEMENTAR EVENTO CLICK EN PRIMERA PREGUNTA

Tarea-07	**Implementar la funcionalidad del botón “volver”**
Descripción	Capturar cuando se pulse el botón “volver” que tienen los dispositivos Android y hacer que se cargue la actividad que nos llevó hasta dónde nos encontramos.
Dependencias	Tarea-06
Estimado:	**1** **Estado:** **TODO**

TABLA 37: TAREA-07: IMPLEMENTAR LA FUNCIONALIDAD DEL BOTÓN "VOLVER"

Tarea-08	**Implementar la actividad “Ponerse chaleco”**
Descripción	Implementar la actividad correspondiente a la pantalla “Ponerse chaleco”.
Dependencias	Tarea-03
Estimado:	**1** **Estado:** **TODO**

TABLA 38: TAREA-08: IMPLEMENTAR LA ACTIVIDAD "PONERSE CHALECO"

Tarea-09	**Implementar paso de pantallas deslizando el dedo**
Descripción	Capturar los eventos que se producen cuando el usuario toca la pantalla y deja de tocarla. Registrar los puntos de la pantalla en los que se ha llevado a cabo cada acción y restar sus coordenadas x para saber si se ha movido de derecha a izquierda o de izquierda a derecha. Cargar la actividad posterior o anterior respectivamente.
Dependencias	Tarea-05 Tarea-08
Estimado:	**2** **Estado: TODO**

TABLA 39: TAREA-09: IMPLEMENTAR PASO DE PANTALLAS DESLIZANDO EL DEDO

Tarea-10	**Añadir efecto al cambiar de pantalla**
Descripción	Incluir un efecto al pasar de una pantalla a otra.
Dependencias	Tarea-07 Tarea-09
Estimado:	**2** **Estado: TODO**

TABLA 40: TAREA-10: AÑADIR EFECTO AL CAMBIAR DE PANTALLA

Tarea-11	**Implementar la actividad “Colocar triángulos”**
Descripción	Implementar la actividad correspondiente a “Colocar triángulos”, enlazándola como posterior de “Ponerse chaleco”.
Dependencias	Tarea-08
Estimado:	**1** **Estado: TODO**

TABLA 41: TAREA-11: IMPLEMENTAR LA ACTIVIDAD "COLOCAR TRIÁNGULOS"

Tarea-12	**Implementar la actividad “Apagar cigarrillo”**
Descripción	Implementar la actividad correspondiente a “Apagar cigarrillo”, enlazándola como posterior de “Colocar triángulos”.
Dependencias	Tarea-11
Estimado:	**1** **Estado: TODO**

TABLA 42: TAREA-12: IMPLEMENTAR LA ACTIVIDAD "APAGAR CIGARRILLO"

Tarea-13	**Implementar la actividad “¿Bien iluminado?”**
Descripción	Implementar la actividad correspondiente a “¿Bien iluminado?”, enlazándola como posterior de “Apagar cigarrillo”.
Dependencias	Tarea-02 Tarea-12
Estimado:	**1** **Estado: TODO**

TABLA 43: TAREA-13: IMPLEMENTAR LA ACTIVIDAD "¿BIEN ILUMINADO?"

Tarea-14	**Implementar la actividad “Ilumine zona”**
Descripción	Implementar la actividad correspondiente a “Ilumine zona”, enlazándola como posterior de “¿Bien iluminado?” en caso de responder a dicha pregunta con un “No”.
Dependencias	Tarea-13
Estimado:	**1** **Estado: TODO**

TABLA 44: TAREA-14: IMPLEMENTAR LA ACTIVIDAD "ILUMINE ZONA"

Tarea-15	**Implementar la actividad “¿Hay incendio?”**
Descripción	Implementar la actividad correspondiente a “¿Hay incendio?”, enlazándola como posterior de “¿Bien iluminado?” en caso de responder a dicha pregunta con un “Sí” y también como posterior de “Ilumine zona”.
Dependencias	Tarea-14
Estimado:	**1** **Estado: TODO**

TABLA 45: TAREA-15: IMPLEMENTAR LA ACTIVIDAD "¿HAY INCENDIO?"

Tarea-16	**Implementar la actividad “Utilizar extintor”**
Descripción	Implementar la actividad correspondiente a “Utilizar extintor”, enlazándola como posterior de “¿Hay incendio?” en caso de responder a dicha pregunta con un “Sí”.
Dependencias	Tarea-15
Estimado:	**1** **Estado: TODO**

TABLA 46: TAREA-16: IMPLEMENTAR LA ACTIVIDAD "UTILIZAR EXTINTOR"

Tarea-17	**Implementar la actividad “Aproximarse al accidente”**
Descripción	Implementar la actividad correspondiente a “Aproximarse al accidente”, enlazándola como posterior de “¿Hay incendio?” en caso de responder a dicha pregunta con un “No” y también como posterior de “Utilizar extintor”.
Dependencias	Tarea-16
Estimado:	**1** **Estado: TODO**

TABLA 47: TAREA-17: IMPLEMENTAR LA ACTIVIDAD "APROXIMARSE AL ACCIDENTE"

Tarea-18	**Implementar la actividad “Desconectar contacto”**
Descripción	Implementar la actividad correspondiente a “Desconectar contacto”, enlazándola como posterior de “Aproximarse al accidente”.
Dependencias	Tarea-17
Estimado:	**1** **Estado: TODO**

TABLA 48: TAREA-18: IMPLEMENTAR LA ACTIVIDAD "DESCONECTE CONTACTO"

Tarea-19	**Implementar la actividad “Avisar autoridades”**
Descripción	Implementar la actividad correspondiente a “Avise autoridades”, enlazándola como posterior de “Desconectar contacto”.
Dependencias	Tarea-18
Estimado:	**1** **Estado: TODO**

TABLA 49: TAREA-19: IMPLEMENTAR LA ACTIVIDAD "AVISAR AUTORIDADES"

- Iteración 2

Tarea-20	**Añadir Google Maps a la actividad “Avisar autoridades”**
Descripción	Incluir un mapa de Google Maps mostrando un marcador en el lugar dónde se encuentre el usuario en el momento de cargar el mapa. Al seleccionar el marcador, se deben mostrar las coordenadas del accidente.
Dependencias	Tarea-19
Estimado:	**4** **Estado: TODO**

TABLA 50: TAREA-20: AÑADIR GOOGLE MAPS A LA ACTIVIDAD "AVISAR AUTORIDADES"

Tarea-21	Implementar la actividad “¿Hay heridos?”
Descripción	Implementar la actividad correspondiente a “¿Hay heridos?”, enlazándola como posterior de “Avisar autoridades”.
Dependencias	Tarea-19
Estimado: 1	Estado: TODO

TABLA 51: TAREA-21: IMPLEMENTAR LA ACTIVIDAD "¿HAY HERIDOS?"

Tarea-22	Implementar la actividad “No mover víctimas”
Descripción	Implementar la actividad correspondiente a “No mover víctimas”, enlazándola como posterior de “¿Hay heridos?” en caso de responder a dicha pregunta con un “Sí”.
Dependencias	Tarea-21
Estimado: 1	Estado: TODO

TABLA 52: TAREA-22: IMPLEMENTAR LA ACTIVIDAD "NO MOVER VÍCTIMAS"

Tarea-23	Implementar la actividad “¿Está consciente?”
Descripción	Implementar la actividad correspondiente a “¿Está consciente?”, enlazándola como posterior de “No mover víctimas”.
Dependencias	Tarea-22
Estimado: 1	Estado: TODO

TABLA 53: IMPLEMENTAR LA ACTIVIDAD "¿ESTÁ CONSCIENTE?"

Tarea-24	Implementar la actividad “¿Tiene heridas?”
Descripción	Implementar la actividad correspondiente a “¿Tiene heridas?”, enlazándola como posterior de “¿Está consciente?” en caso de responder a dicha pregunta con un “Sí”.
Dependencias	Tarea-23
Estimado: 1	Estado: TODO

TABLA 54: TAREA-24: IMPLEMENTAR LA ACTIVIDAD "¿TIENE HERIDAS?"

Tarea-25	Implementar la actividad “¿Hemorragia incontrolable?”
Descripción	Implementar la actividad correspondiente a “¿Hemorragia incontrolable?”, enlazándola como posterior de “¿Tiene heridas?” en caso de responder a dicha pregunta con un “Sí”.
Dependencias	Tarea-24
Estimado: 1	Estado: TODO

TABLA 55: TAREA-25: IMPLEMENTAR LA ACTIVIDAD "¿HEMORRAGIA INCONTROLABLE?"

Tarea-26	Implementar la actividad “Aplicar torniquete”
Descripción	Implementar la actividad correspondiente a “Aplicar torniquete”, enlazándola como posterior de “¿Hemorragia incontrolable?” en caso de responder a dicha pregunta con un “Sí”. Inicialmente esta actividad incluirá una imagen, que posteriormente será sustituida por un vídeo.
Dependencias	Tarea-25
Estimado: 1	Estado: TODO

TABLA 56: TAREA-26: IMPLEMENTAR LA ACTIVIDAD "APLICAR TORNIQUETE"

Tarea-27	Implementar la actividad “Presionar herida”
Descripción	Implementar la actividad correspondiente a “Presionar herida”, enlazándola como posterior de “¿Hemorragia incontrolable?” en caso de responder a dicha pregunta con un “No”. Inicialmente esta actividad incluirá una imagen, que posteriormente será sustituida por un vídeo.
Dependencias	Tarea-25
Estimado: 1	Estado: TODO

TABLA 57: TAREA-27: IMPLEMENTAR LA ACTIVIDAD "PRESIONAR HERIDA"

Tarea-28	**Implementar la actividad “Abrigar herido”**
Descripción	Implementar la actividad correspondiente a “Abrigar herido”, enlazándola como posterior de “Aplicar torniquete” y también como posterior de “Presionar herida”.
Dependencias	Tarea-26 Tarea-27
Estimado:	**1** **Estado:** **TODO**

TABLA 58: TAREA-28: IMPLEMENTAR LA ACTIVIDAD "ABRIGAR HERIDO"

Tarea-29	**Implementar la actividad “No ingerir nada”**
Descripción	Implementar la actividad correspondiente a “No ingerir nada”, enlazándola como posterior de “Abrigar herido”.
Dependencias	Tarea-28
Estimado:	**1** **Estado:** **TODO**

TABLA 59: TAREA-29: IMPLEMENTAR LA ACTIVIDAD "NO INGERIR NADA"

Tarea-30	**Implementar la actividad “Espere autoridades”**
Descripción	Implementar la actividad correspondiente a “Espere autoridades”, enlazándola como posterior de “¿Hay heridos?” en caso de responder a dicha pregunta con un “No” y también como posterior de “No ingerir nada”. Esta actividad incluirá dos botones, como si de una pregunta se tratará. El primero de ellos deberá redirigir a la actividad inicial de la aplicación, mientras que el segundo deberá salir de la aplicación.
Dependencias	Tarea-29
Estimado:	**3** **Estado:** **TODO**

TABLA 60: TAREA-30: IMPLEMENTAR LA ACTIVIDAD "ESPERE AUTORIDADES"

Tarea-31	**Implementar la actividad “¿Está herido?”**
Descripción	Implementar la actividad correspondiente a “¿Está herido?”, enlazándola como posterior de la actividad inicial de la aplicación en caso de seleccionar la opción “He tenido un accidente”.
Dependencias	Tarea-04
Estimado:	**1** **Estado:** **TODO**

TABLA 61: TAREA-31: IMPLEMENTAR LA ACTIVIDAD "¿ESTÁ HERIDO?"

Tarea-32	**Añadir enlace a la actividad “Avisar autoridades”**
Descripción	Añadir enlace a la actividad “Avisar autoridades” para que se incluya como su anterior “¿Está herido?” en caso de venir por el camino correspondiente a la respuesta “He tenido un accidente” en la actividad inicial y la respuesta “Sí” en la actividad correspondiente a “Está herido”.
Dependencias	Tarea-19 Tarea-31
Estimado:	**1** **Estado:** **TODO**

TABLA 62: TAREA-32: AÑADIR ENLACE A LA ACTIVIDAD "AVISAR AUTORIDADES"

Tarea-33	**Añadir enlace a la actividad “Estacionar vehículo”**
Descripción	Añadir enlace a la actividad “Estacionar vehículo” para que se incluya como su anterior “¿Está herido?” en caso de venir por el camino correspondiente a la respuesta “He tenido un accidente” en la actividad inicial.
Dependencias	Tarea-05 Tarea-31
Estimado:	**1** **Estado:** **TODO**

TABLA 63: TAREA-33: AÑADIR ENLACE A LA ACTIVIDAD "ESTACIONAR VEHÍCULO"

Tarea-34	**Añadir enlace a la actividad “Colocar triángulos”**
Descripción	Añadir enlace a la actividad “Colocar triángulos” para que se incluya como su siguiente “Avisar autoridades” en caso de venir por el camino correspondiente a la respuesta “He tenido un accidente” en la actividad inicial.
Dependencias	Tarea-19
Estimado:	**1** **Estado:** **TODO**

TABLA 64: TAREA-34: AÑADIR ENLACE A LA ACTIVIDAD "COLOCAR TRIÁNGULOS"

Tarea-35	**Añadir enlace a la actividad “Avisar autoridades”**
Descripción	Añadir enlace a la actividad “Avisar autoridades” para que se incluya como su anterior “Colocar triángulos” en caso de venir por el camino correspondiente a la respuesta “He tenido un accidente” en la actividad inicial y la respuesta “No” en la actividad correspondiente a “Está herido”.
Dependencias	Tarea-19 Tarea-31
Estimado:	**1** **Estado:** **TODO**

TABLA 65: TAREA-35: AÑADIR ENLACE A LA ACTIVIDAD "AVISAR AUTORIDADES"

Tarea-36	**Implementar la actividad “¿Guardar datos?”**
Descripción	Implementar la actividad correspondiente a “¿Guardar datos?”, enlazándola como posterior de “Avisar autoridades” en caso de venir por el camino correspondiente a la respuesta “He tenido un accidente en la actividad inicial”.
Dependencias	Tarea-35
Estimado:	**1** **Estado:** **TODO**

TABLA 66: TAREA-36: IMPLEMENTAR LA ACTIVIDAD "¿GUARDAR DATOS?"

Tarea-37	**Diseño de las pantallas de datos**
Descripción	Realizar el diseño visual de las pantallas que vayan a servir para introducir los datos del accidente y del asegurado, así como para visualizar esos datos a posteriori.
Dependencias	Tarea-01
Estimado:	**3** **Estado: TODO**

TABLA 67: TAREA-37: DISEÑO DE LAS PANTALLAS DE DATOS

Tarea-38	**Implementar la actidad “Datos accidente”**
Descripción	Implementar la actividad correspondiente a “Datos accidente”, enlazándola como posterior de “¿Guardar datos?” en caso de responder a dicha pregunta con un “Sí”. Los datos introducidos en esta pantalla deberán propagarse al siguiente paso.
Dependencias	Tarea-36
Estimado:	**1** **Estado: TODO**

TABLA 68: TAREA-38: IMPLEMENTAR LA ACTIVIDAD "DATOS ACCIDENTE"

Tarea-39	**Implementar la actividad “Datos asegurado”**
Descripción	Implementar la actividad correspondiente a “Datos asegurado”, enlazándola como posterior de “Datos accidente”.
Dependencias	Tarea-38
Estimado:	**1** **Estado: TODO**

TABLA 69: TAREA-39: IMPLEMENTAR LA ACTIVIDAD "DATOS ASEGURADO"

Tarea-40	**Implentar la actividad “Fin”**
Descripción	Implementar la actividad correspondiente a “Fin”, enlazándola como posterior de “Avisar autoridades” en caso venir por el camino correspondiente a la respuesta “He tenido un accidente” en la actividad inicial, como posterior de “¿Guardar datos?” en caso de responder a dicha pregunta con un “No” y también como posterior de “Datos asegurado”. Esta actividad incluirá dos botones, como si de una pregunta se tratará. El primero de ellos deberá redirigir a la actividad inicial de la aplicación, mientras que el segundo deberá salir de la aplicación.
Dependencias	Tarea-39
Estimado:	**3** **Estado:** **TODO**

TABLA 70: TAREA-40: IMPLEMENTAR LA ACTIVIDAD "FIN"

- Iteración 3

Tarea-41	**Diseño de la base de datos**
Descripción	Realizar el diseño de la base de datos con SQLite y crear la primera versión de la misma.
Dependencias	Tarea-01
Estimado:	**2** **Estado:** **TODO**

TABLA 71: TAREA-41: DISEÑO DE LA BASE DE DATOS

Tarea-42	**Implementar el grabado de datos**
Descripción	Implementar la funcionalidad para guardar los datos en la base de datos a partir de la información introducida por el usuario en las pantallas de “Datos accidente” y “Datos asegurado”. El guardado deberá hacerse al pulsar el botón guardar de “Datos asegurado”.
Dependencias	Tarea-39 Tarea-41
Estimado:	**2** **Estado:** **TODO**

TABLA 72: TAREA-42: IMPLEMENTAR EL GRABADO DE DATOS

Tarea-43	**Utilizar la cámara de fotos para hacer fotos**
Descripción	Añadir la funcionalidad para poder utilizar la cámara de fotos del dispositivo cuando el usuario pulse el botón “Tomar fotos” de las pantallas “Datos accidente” y “Datos asegurado”. Una vez hecha la foto se volverá a la pantalla desde la que se inició la cámara.
Dependencias	Tarea-39
Estimado:	**4** **Estado:** **TODO**

TABLA 73: TAREA-43: UTILIZAR LA CÁMARA DE FOTOS PARA HACER FOTOS

Tarea-44	**Implementar la actividad “Datos guardados”**
Descripción	Implementar la actividad que se lanzará cuando el usuario seleccione la opción “Datos guardados” del menú. Esta actividad deberá leer todos los registros de accidentes guardados en la base de datos y creará un botón por cada uno para que el usuario escoja cual quiere visualizar. El botón contendrá como texto el almacenado en el campo fecha del registro.
Dependencias	Tarea-41
Estimado:	**2** **Estado:** **TODO**

TABLA 74: TAREA-44: IMPLEMENTAR LA ACTIVIDAD "DATOS GUARDADOS"

Tarea-45	**Cargar datos de un accidente desde base de datos**
Descripción	Cuando el usuario seleccione un accidente en la pantalla “Datos guardados”, se accederá a toda la información de ese accidente y se cargará en las pantallas de “Datos accidente” y “Datos asegurado”.
Dependencias	Tarea-44
Estimado:	**2** **Estado:** **TODO**

TABLA 75: TAREA-45: CARGAR DATOS DE UN ACCIDENTE DESDE BASE DE DATOS

Tarea-46	**Cargar fotos de un accidente**
Descripción	Cuando el usuario esté visualizando los datos de un accidente guardados previamente, en lugar de mostrarse los botones "Tomar fotos" y "Guardar" de las pantallas "Datos accidente" y "Datos asegurado", se mostrará un único botón para ver las fotos realizadas a un accidente. Este botón abrirá la carpeta dónde se guardaron las fotos tomadas y cuando se seleccione una de ellas se mostrará al usuario en una nueva pantalla.
Dependencias	Tarea-43
Estimado:	**3** **Estado:** **TODO**

TABLA 76: TAREA-46: CARGAR FOTOS DE UN ACCIDENTE

Tarea-47	**Diseño de la vista del botón "menú"**
Descripción	Diseñar los elementos que compondrán el menú disponible cuando se pulse el botón "Menú" disponible en los dispositivos Android.
Dependencias	Tarea-01
Estimado:	**1** **Estado:** **TODO**

TABLA 77: TAREA-47: DISEÑO DE LA VISTA DEL BOTÓN "MENÚ"

Tarea-48	**Añadir "menú" a las actividades**
Descripción	Incluir en todas las actividades el evento que cargue el menú definido cuando se pulse el botón "Menú".
Dependencias	Tarea-40 Tarea-44
Estimado:	**3** **Estado:** **TODO**

TABLA 78: TAREA-48: AÑADIR "MENÚ" A LAS ACTIVIDADES

Tarea-49	Implementar la funcionalidad al seleccionar un menú
Descripción	Implementar los eventos que capturen que opción del menú se ha seleccionado y redirigir a la pantalla correspondiente en cada una.
Dependencias	Tarea-02 Tarea-44
Estimado:	2 Estado: TODO

TABLA 79: TAREA-49: IMPLEMENTAR LA FUNCIONALIDAD AL SELECCIONAR UN MENÚ

Tarea-50	Añadir audio al iniciar una pantalla
Descripción	Incluir un reproductor de audio en cada pantalla que reproduzca un archivo de audio concreto según la pantalla que estemos iniciando.
Dependencias	Tarea-40
Estimado:	5 Estado: TODO

TABLA 80: TAREA-50: AÑADIR AUDIO AL INICIAR UNA PANTALLA

Tarea-51	Finalizar reproducción de audio al cambiar de pantalla
Descripción	Evitar que se reproduzca un audio mientras se muestra una pantalla que no es la suya. Para ello, si se cambia de pantalla mientras el audio no ha finalizado, se forzará la parada de la reproducción antes de cargar la siguiente pantalla.
Dependencias	Tarea-50
Estimado:	2 Estado: TODO

TABLA 81: TAREA-51: FINALIZAR REPRODUCCIÓN DE AUDIO AL CAMBIAR DE PANTALLA

Tarea-52	Añadir funcionalidad de repetir audio a las actividades
Descripción	Añadir un botón para repetir el audio de la pantalla en aquellas que representen acciones que debe realizar el usuario.
Dependencias	Tarea-50
Estimado:	2 Estado: TODO

TABLA 82: TAREA-52: AÑADIR FUNCIONALIDAD DE REPETIR AUDIO A LAS ACTIVIDADES

- Iteración 4

Tarea-53	Añadir reproducción de vídeo a “Aplicar torniquete”
Descripción	Incluir un reproductor de vídeo en la pantalla “Aplicar torniquete”. Este reproductor deberá incluir los botones estándar de reproducción para poder pausar, reanudar la reproducción, etc.
Dependencias	Tarea-26
Estimado: 5	Estado: TODO

TABLA 83: TAREA-53: AÑADIR REPRODUCCIÓN DE VÍDEO A "APLICAR TORNIQUETE"

Tarea-54	Añadir reproducción de vídeo a “Presionar herida”
Descripción	Incluir un reproductor de vídeo en la pantalla “Presionar herida”. Este reproductor deberá incluir los botones estándar de reproducción para poder pausar, reanudar la reproducción, etc.
Dependencias	Tarea-27 Tarea-53
Estimado: 2	Estado: TODO

TABLA 84: TAREA-54: AÑADIR REPRODUCCIÓN DE VÍDEO A "PRESIONAR HERIDA"

Tarea-55	Finalizar reproducción de vídeo al cambiar de pantalla
Descripción	Evitar que se reproduzca un vídeo mientras se muestra una pantalla que no es la suya. Para ello, si se cambia de pantalla cuando el vídeo sigue en reproducción, se forzará la parada de la reproducción antes de cargar la siguiente pantalla.
Dependencias	Tarea-54
Estimado: 1	Estado: TODO

TABLA 85: TAREA-55: FINALIZAR REPRODUCCIÓN DE VÍDEO AL CAMBIAR DE PANTALLA

Tarea-56	**Añadir detalles a los diseños**
Descripción	Incluir detalles gráficos a los distintos elementos de la aplicación: botones con esquinas redondeadas, flechas en las pantallas que indiquen si se puede ir a la siguiente pantalla o a la anterior y otros retoques visuales.
Dependencias	Tarea-02 Tarea-03 Tarea-37 Tarea-47
Estimado: 6	**Estado: TODO**

TABLA 86: AÑADIR DETALLES A LOS DISEÑOS

4

DISEÑO DE LA APLICACIÓN

4. DISEÑO DE LA APLICACIÓN

4.1. ARQUITECTURA

La aplicación cuenta con una arquitectura sencilla, ya que únicamente se conecta a una base de datos interna del propio dispositivo para almacenar los datos de los accidentes, por lo que tendremos una estructura similar a:

FIGURA 39: ARQUITECTURA DE LA APLICACIÓN

No obstante, pese a tener una arquitectura sencilla, hay que tener en cuenta que para el correcto funcionamiento de la aplicación, es necesario dar a la aplicación permisos especiales sobre el dispositivo en el que estará instalada. Esta lista de permisos incluye:

- Acceso a internet: Necesario para poder cargar los mapas de Google Maps, así como para obtener la ubicación mediante la red (por si no tenemos GPS o este no está activo).
- Acceso al GPS: Necesario para poder obtener la ubicación mediante el GPS si este está encendido.
- Permiso para hacer llamadas: Necesario para poder realizar llamadas a los servicios de emergencias.
- Permiso para utilizar la cámara: Necesario para poder hacer fotos del accidente.

Todos estos permisos se definirán en el archivo *AndroidManifest.xml*.

4.2. DISEÑO INICIAL

El diseño inicial de la aplicación nos va a permitir definir qué soluciones vamos a implementar para cumplir con los requisitos definidos previamente. Este diseño se va a dividir en tres partes. La primera de ellas dedicada al diseño de las pantallas de la aplicación, la segunda al diseño de la base de datos y la tercera a decisiones de

implementación. Cada propuesta identificará una solución que será presentada en una tabla con la siguiente estructura:

Código: Nombre	
Descripción	

TABLA 87: FORMATO TABLAS DE SOLUCIONES

- Código: Incluirá la cadena SOL seguida de un número. Este número comenzará en el 1 e irá incrementándose en los sucesivos requisitos.
- Nombre: Nombre que dé una idea clara del objetivo de la solución propuesta.
- Descripción: Incluirá una descripción completa de la solución.

➢ Diseño de las pantallas

Para la correcta presentación de los requisitos, es necesario diseñar las pantallas que se mostrarán al usuario de la forma más cómoda posible. Estás pantallas se dividirán en las siguientes soluciones:

SOL-01: Diseño de la pantalla de preguntas	
Descripción	Se dividirá en dos secciones. La primera de ellas contendrá el texto de la pregunta, mientras que la segunda contendrá las opciones a escoger por el usuario.

TABLA 88: SOL-01: DISEÑO DE LA PANTALLA DE PREGUNTAS

SOL-02: Pantalla de acciones	
Descripción	Se dividirá en tres secciones. La primera contendrá un texto descriptivo de la acción a realizar. A continuación tendremos una imagen descriptiva de la acción o un video, en caso de que sea necesario para aclarar la acción. Finalmente aparecerá un botón junto con un texto para volver a reproducir el audio asociado a la acción.

TABLA 89: SOL-02: PANTALLA DE ACCIONES

SOL-03: Pantalla de ubicación del accidente	
Descripción	Esta pantalla se dividirá en tres partes. En la primera de ellas aparecerá un texto indicando la acción a realizar. A continuación se dispondrá de un botón para llamar directamente al teléfono de emergencias. Finalmente se mostrará un mapa indicando la posición en la que se encuentre el usuario.

TABLA 90: SOL-03: PANTALLA DE UBICACIÓN DEL ACCIDENTE

SOL-04: Pantalla de datos	
Descripción	Esta pantalla permitirá tanto introducir datos relacionados con el accidente, como visualizarlos posteriormente. Se dividirá en cuatro partes. La primera de ellas, contendrá un texto indicando los datos que se van a guardar o visualizar (datos del accidente o datos del asegurado). A continuación, vendrá un texto junto con un cuadro de texto pequeño, para introducir o mostrar un dato concreto. Debajo de esto, se ubicará un cuadro de texto más grande, donde se introducirá o mostrará información más larga y finalmente se tendrán unos botones. En el caso de que se estén ingresando datos, los botones servirán para tomar fotos y para guardar los datos, mientras que en el caso de visualizar los datos, sólo aparecerá un botón para visualizar las fotos del accidente.

TABLA 91: SOL-04: PANTALLA DE DATOS

SOL-05: Pantalla de accidentes almacenados	
Descripción	Para poder visualizar los datos de un accidente previo, se mostrará una pantalla en la cual aparecerán los distintos accidentes de los que se tienen datos, uno a continuación de otro.

TABLA 92: SOL-05: PANTALLA DE ACCIDENTES ALMACENADOS

A continuación pueden verse los bocetos de todas las pantallas citadas anteriormente:

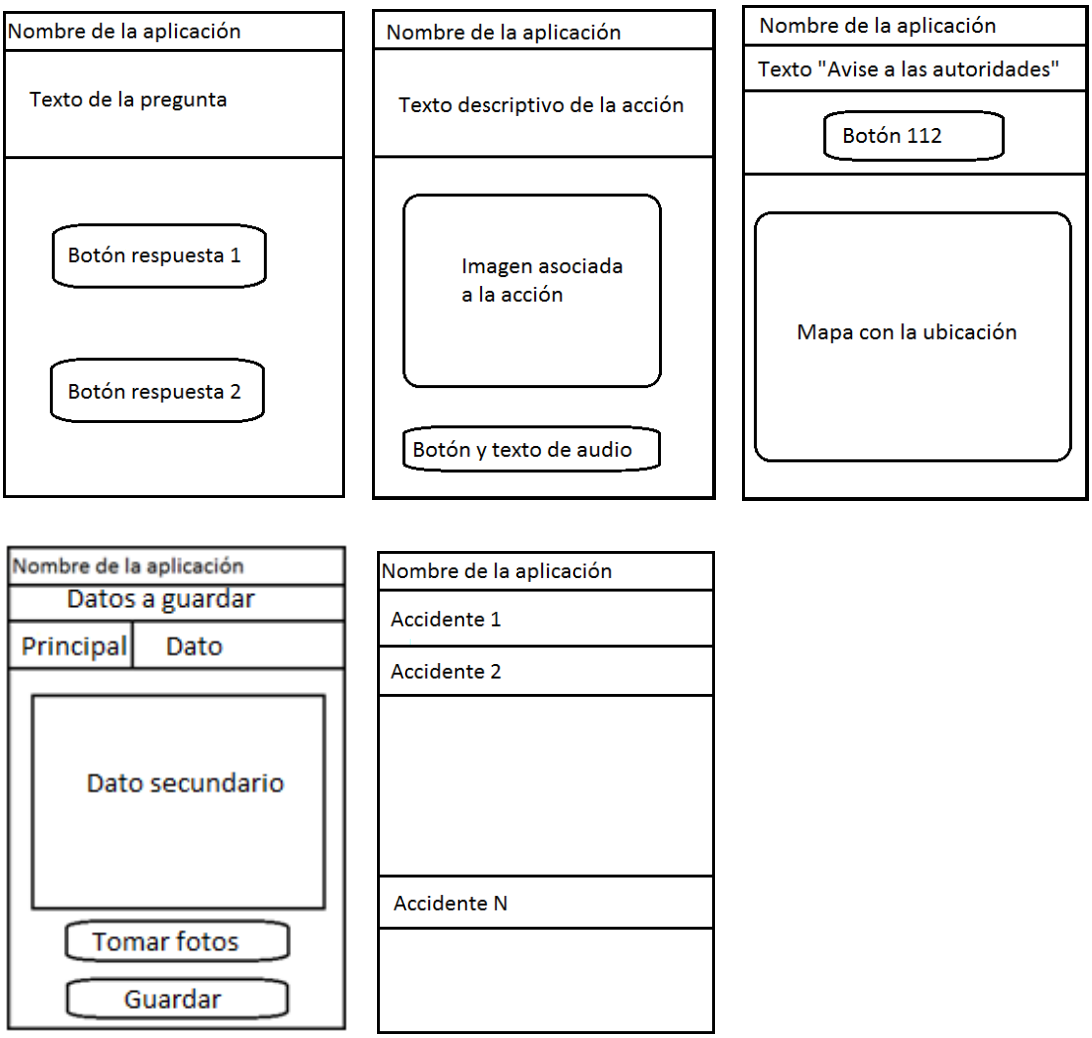


FIGURA 40: BOCETOS DEL DISEÑO INICIAL DE LAS PANTALLAS

- Base de datos

Para poder almacenar la información de varios accidentes, es necesario que contemos con un entorno de persistencia para los datos:

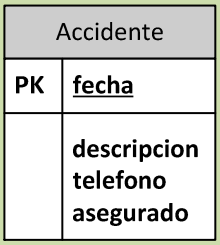
SOL-06: Persistencia de datos	
Descripción	Se utilizará una base de datos interna, que almacenará la información relativa a los accidentes, y que será proporcionada por el usuario en las pantallas correspondientes. Esta base de datos contará con una única tabla, que tendrá la siguiente estructura:  Se almacenará como campo identificativo la fecha del accidente. Además de este dato, se almacenará un campo por cada elemento visual mostrado al usuario para introducir datos. De esta forma, se tendrá un campo para almacenar la descripción introducida en los datos del accidente, otro para almacenar el teléfono de los datos del asegurado y otro para almacenar la información del asegurado.

TABLA 93: SOL-06: PERSISTENCIA DE DATOS

SOL-07: Recuperación de datos	
Descripción	Se accederá a la base de datos buscando el registro cuya fecha coincida con la fecha indicada en el botón pulsado por el usuario en la pantalla de accidentes almacenados.

TABLA 94: SOL-07: RECUPERACIÓN DE DATOS

- Decisiones de implementación

SOL-08: Capturar eventos cuando se pulse la pantalla	
Descripción	Se capturarán los eventos relacionados con la pulsación de la pantalla y cuando se deje de pulsar. De esta forma, si vemos que el dedo se ha movido de derecha a izquierda o de izquierda a derecha, sabremos que debemos mostrar el paso siguiente o anterior respectivamente.

TABLA 95: SOL-08: CAPTURAR EVENTOS CUANDO SE PULSE LA PANTALLA

➢ Relación requisitos funcionales-soluciones

Para comprobar la cobertura que nuestras soluciones van a tener sobre los requisitos, a continuación podemos ver qué requisitos funcionales se van a ver afectados por las soluciones que se han propuesto:

	SOL-01	SOL-02	SOL-03	SOL-04	SOL-05	SOL-06	SOL-07	SOL-08
RF-01		X						
RF-02		X						
RF-03		X						
RF-04		X						
RF-05								X
RF-06		X						
RF-07		X						
RF-08								X
RF-09	X							
RF-10			X					
RF-11			X					
RF-12			X					
RF-13				X				
RF-14				X				
RF-15				X				
RF-16					X	X	X	
RF-17								
RF-18								X

TABLA 96: RELACIÓN REQUISITOS FUNCIONALES-SOLUCIONES

4.3. DISEÑO FINAL

Como se comentó a la hora de hacer el diseño conceptual, vamos a tener que definir una serie de interfaces de usuario que sirvan como punto de entrada a la aplicación. Para llevarlos a cabo vamos a contar con varios archivos *xml* genéricos que definirán aspectos comunes que podrán ser utilizados por los diseños. Estos archivos son:

- *strings.xml*: Define las cadenas de texto que se utilizarán en las distintas pantallas.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <!-- Genéricos -->
    <string name="app_name">Asistencia en carretera</string>
    <string name="texto_repetir_audio">Repetir Audio</string>
    <string name="respuesta_si">SÍ</string>
    <string name="respuesta_no">NO</string>

    <!-- Menu -->
    <string name="menu_sos">S.O.S</string>
    <string name="menu_datos_guardados">Datos guardados</string>
    <string name="menu_ayuda">Ayuda</string>

    <!-- tipo_incidencia -->
    <string name="pregunta_tipo_incidencia">¿QUÉ TIPO DE INCIDENCIA HA OCURRIDO?</string>
    <string name="respuesta_visto_accidente">HE VISTO un accidente</string>
    <string name="respuesta_tenido_accidente">HE TENIDO un accidente</string>

    **** resto de cadenas ****

</resources>
```

FIGURA 41: DISEÑO FINAL: STRINGS.XML

- *styles.xml*: Define los estilos que se utilizarán en las distintas pantallas. Estos estilos incluyen características como tamaños, márgenes, colores, etc.

```
<resources xmlns:android="http://schemas.android.com/apk/res/android">

    <style name="FondoPrincipal">
        <item name="android:layout_width">match_parent</item>
        <item name="android:layout_height">match_parent</item>
        <item name="android:background">#FFFEE9</item>
    </style>

    <style name="TextoPrincipal">
        <item name="android:layout_width">match_parent</item>
        <item name="android:layout_height">96dp</item>
        <item name="android:gravity">center</item>
        <item name="android:textColor">#000000</item>
        <item name="android:textSize">20sp</item>
    </style>

    <style name="ImagenPrincipal">
        <item name="android:layout_width">240dp</item>
        <item name="android:layout_height">240dp</item>
        <item name="android:layout_gravity">center</item>
    </style>

    **** resto de estilos ****

</resources>
```

FIGURA 42: DISEÑO FINAL: STYLES.XML

- *boton_general.xml* y *boton_112.xml*: Definen algunas características propias de los botones que se utilizarán para incluir las respuestas a las preguntas y el botón de llamada a los servicios de emergencias.

```xml
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true" >
        <shape>
            <solid
                android:color="#777777" />
            <stroke
                android:width="1dp"
                android:color="#222222" />
            <corners
                android:radius="3dp" />
            <padding
                android:left="10dp"
                android:top="10dp"
                android:right="10dp"
                android:bottom="10dp" />
        </shape>
    </item>
    <item>
        <shape>
            <gradient
                android:startColor="#777777"
                android:endColor="#222222"
                android:angle="270" />
            <stroke
                android:width="1dp"
                android:color="#222222" />
            <corners
                android:radius="10dp" />
            <padding
                android:left="10dp"
                android:top="10dp"
                android:right="10dp"
                android:bottom="10dp" />
        </shape>
    </item>
</selector>
```

FIGURA 43: DISEÑO FINAL: BOTONES

Una vez definidas las características comunes, cada tipo de pantalla tendrá su propia estructura que hará uso de una o varias de esas características previamente definidas.

En primer lugar tendremos el diseño de las pantallas correspondientes a preguntas. Como se vio en el diseño conceptual, esta pantalla consta de un texto correspondiente a la pregunta y dos botones con las posibles respuestas a escoger por el usuario.

```
<?xml version="1.0" encoding="utf-8"?>
  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:id="@+id/vistaTipoIncidencia"
     style="@style/FondoPrincipal"
     android:orientation="vertical" >

     <TextView
        style="@style/TextoPrincipal"
        android:text="@string/pregunta_tipo_incidencia" />

     <Button
        android:id="@+id/botonVistoAccidente"
        style="@style/BotonGeneral"
        android:text="@string/respuesta_visto_accidente" />

     <Button
        android:id="@+id/botonTenidoAccidente"
        style="@style/BotonGeneral"
        android:text="@string/respuesta_tenido_accidente" />

  </LinearLayout>
```

FIGURA 44: DISEÑO FINAL: PANTALLA DE PREGUNTA

Como puede verse en el código tenemos definido un *TextView* para el texto de la pregunta y dos *Button* para las correspondientes respuestas, todo ello incluido en un *LinearLayout* que dispone los elementos en vertical, es decir, cada uno debajo del anterior.

Una vez diseñadas las pantallas de las preguntas, pasamos al diseño de las correspondientes a las acciones. Tal y como se especificó, estas pantallas constarán de un texto, una imagen (o un vídeo) y un botón para repetir el audio de la pantalla.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/vistaEstacionarVehiculo"
   style="@style/FondoPrincipal"
   android:orientation="vertical" >

   <TextView
      style="@style/TextoPrincipal"
      android:text="@string/texto_estacionar_vehiculo" />

   <!-- Layout con las flechas y la imagen -->
   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:orientation="horizontal" >

      <ImageView
         style="@style/FlechalzquierdaImagen" />

    <ImageView
       android:id="@+id/estacionarImg"
       style="@style/ImagenPrincipal"
       android:contentDescription="@string/texto_estacionar_vehiculo"
       android:src="@drawable/estacionar_img" />

    <ImageView
         style="@style/FlechaDerechaImagen" />
  </LinearLayout>

   <!-- Layout repetir audio -->
   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="horizontal" >

      <ImageView
         android:id="@+id/estacionarVehiculoAudio"
         style="@style/BotonRepetirAudio" />

      <TextView
         style="@style/TextoRepetir"
         android:text="@string/texto_repetir_audio" />
   </LinearLayout>

</LinearLayout>
```

FIGURA 45: DISEÑO FINAL: PANTALLA DE ACCIÓN

En el código vemos claramente las tres partes en las que se divide la pantalla. La primera con un *TextView* con el texto de la acción, seguida de una sección con tres elementos *ImageView* colocados horizontalmente que definen tres imágenes, la flecha izquierda para indicar que podemos volver hacia atrás, la imagen principal que define la acción y la flecha derecha que indica que podemos avanzar. Finalmente tenemos otra sección con dos elementos colocados horizontalmente, uno correspondiente a un *ImageView* que define la imagen con el icono de repetir y el *TextView* con el texto de "Repetir Audio".

La pantalla correspondiente al mapa tiene aspectos tanto de las pantallas de las preguntas, ya que incluye un botón para llamar a los servicios de emergencias, como de las pantallas de acciones, ya que incluye el mapa de forma parecida a las imágenes en esas pantallas.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/vistaAviseAutoridades"
    style="@style/FondoPrincipal"
    android:orientation="vertical" >

    <TextView
        style="@style/TextoPrincipal"
        android:text="@string/texto_avise_autoridades" />

    <!-- Layout con las flechas y el boton de emergencias -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="-20dp"
        android:gravity="center"
        android:orientation="horizontal" >

        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="left|center"
            android:layout_marginRight="30dp"
            android:src="@drawable/flecha_izq" />

        <Button
            android:id="@+id/botonLlamada"
            style="@style/BotonEmergencias"
            android:background="@drawable/boton_112" />

        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="right|center"
            android:layout_marginLeft="30dp"
            android:src="@drawable/flecha_der" />

    </LinearLayout>

    <TextView
        style="@style/TextoLocalizacion"
        android:text="@string/texto_lugar_accidente" />

    <fragment
        xmlns:map="http://schemas.android.com/apk/res-auto"
        android:id="@+id/mapaLocalizacion"
        android:name="com.google.android.gms.maps.SupportMapFragment"
        style="@style/ImagenMapa" />

</LinearLayout>
```

FIGURA 46: DISEÑO FINAL: PANTALLA CON MAPA

En este caso volvemos a tener un *TextView* con el texto correspondiente a la pantalla, seguido de una sección con tres elementos colocados horizontalmente. Estos elementos se corresponden con las flechas para indicar que se puede retroceder y avanzar, ambas como *ImageView*, que flanquean el *Button* que nos sirve para llamar a los servicios de emergencias. Finalmente tenemos un *TextView* que nos muestra el texto “Lugar del accidente” seguido de un elemento *Fragment* que contendrá la vista de Google Maps.

La siguiente pantalla a comentar corresponde al diseño para introducir los datos del accidente así como los del asegurado en caso de que haya un contrario. Esta pantalla contara con un texto, un primer dato característico de la pantalla de datos, otro dato dónde poder incluir más información y finalmente unos botones.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/vistaDatosAccidente"
    style="@style/FondoPrincipal"
    android:layout_width="match_parent"
    android:orientation="vertical" >

    <TextView
        style="@style/Titulo"
        android:layout_width="match_parent"
        android:text="@string/titluo_datos_accidente" />

    <!-- Bloque fecha -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <TextView
            style="@style/EtiquetaCuadroTextoPequenyo"
            android:text="@string/texto_fecha_hora" />
        <EditText
            android:id="@+id/fechaHora"
            style="@style/CuadroTextoPequenyo"
            android:text="" />
    </LinearLayout>

    <!-- Bloque lugar -->

    <!-- Bloque descripcion -->
    <TextView
        style="@style/EtiquetaCuadroTextoGrande"
        android:text="@string/texto_descripcion" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:orientation="horizontal" >
        <ImageView
            style="@style/FlechaIzquierdaImagen" />
        <EditText
            android:id="@+id/descripcion"
            style="@style/CuadroTextoGrande"
            android:text="" />
        <TextView style="@style/ImagenFlecha" />
    </LinearLayout>

    <!-- Bloque check contrario -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <TextView
            style="@style/EtiquetaCuadroTextoPequenyo"
            android:text="Con contrario"/>
        <CheckBox
            android:id="@+id/contrario"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
    </LinearLayout>

    <!-- Botones -->
</LinearLayout>
```

FIGURA 47: DISEÑO FINAL: DATOS DEL ACCIDENTE

Vemos el *TextView* que identifica que datos estamos guardando, seguido de un grupo compuesto por otro *TextView*, que indica el dato principal a rellenar en esa pantalla, seguido de un *EditText* que nos permite introducir el texto oportuno. A continuación, aparecería el bloque con los datos del lugar, que sería similar al anterior. Posteriormente tenemos otro *TextView* que indica que se espera rellenar en el siguiente campo. A continuación tenemos otro grupo, compuesto por una flecha como un *ImageView* que indicará que podemos volver al paso anterior y otro *EditText* para introducir la

información. Finalmente, tendremos dos *Button* que nos permitirán tomar fotos y guardar los datos introducidos en la base de datos.

La pantalla correspondiente a los datos de asegurado, se ha modificado con respecto a lo diseñado originalmente. Se ha visto que es necesario incluir más información sobre el contrario. Debido a esto se ha realizado un diseño en el cual únicamente aparecen datos a rellenar sencillos. Además, el botón de tomar fotos desaparece, ya que ya se ha mostrado en la pantalla anterior, para dejar únicamente el botón para guardar los datos.

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content">

  <LinearLayout
      android:id="@+id/vistaDatosAsegurado"
      style="@style/FondoPrincipal"
      android:orientation="vertical">

      <TextView
          style="@style/Titulo"
          android:text="@string/titluo_datos_asegurado" />

      <!-- Bloque nombre -->
      <LinearLayout
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:orientation="horizontal" >

          <TextView
              style="@style/EtiquetaCuadroTextoPequenyo"
              android:text="Nombre" />

          <EditText
              android:id="@+id/nombre"
              style="@style/CuadroTextoPequenyo"
              android:text="" />
      </LinearLayout>

      <!-- Bloque nif -->

      <!-- Bloque domicilio -->

      <!-- Bloque telefono -->

      <!-- Bloque matricula -->

      <!-- Bloque marca -->

      <!-- Bloque aseguradora -->

      <!-- Bloque poliza -->

      <Button
          android:id="@+id/guardarDatosAsegurado"
          style="@style/BotonGeneralDatos"
          android:text="@string/boton_guardar" />

  </LinearLayout>
</ScrollView>
```

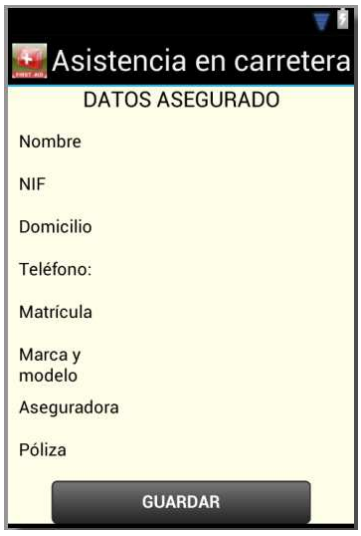


FIGURA 48: DISEÑO FINAL: DATOS DEL ASEGURADO

En este caso, todos los bloques van a contar con un *TextView* que indique el dato que se espera y un *EditText* para introducirlo. Puede verse además que el *LinearLayout* principal de todas las pantallas está ubicado dentro de un *ScrollView*. Esto es así ya que

los datos pueden no caber en la pantalla del dispositivo, por lo que habrá que desplazarla hacia abajo para poder rellenar todos.

Una vez que tenemos los datos de accidentes guardados, para poder recuperarlos diseñamos la pantalla que nos mostrará los accidentes guardados para seleccionar cual queremos visualizar. Esta pantalla contendrá únicamente botones que se crearán dinámicamente en función de la información almacenada en la base de datos. La estructura final de esta pantalla, una vez creados los botones, sería:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/vistaDatosGuardados"
    style="@style/FondoPrincipal"
    android:orientation="vertical" >

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="#black"
        android:layout_marginLeft="20dp"
        android:layout_marginTop="10dp"
        android:layout_marginRight="20dp"
        android:text="10/05/2013 09:37" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="#black"
        android:layout_marginLeft="20dp"
        android:layout_marginTop="10dp"
        android:layout_marginRight="20dp"
        android:text="23/10/2013 16:02" />

</LinearLayout>
```

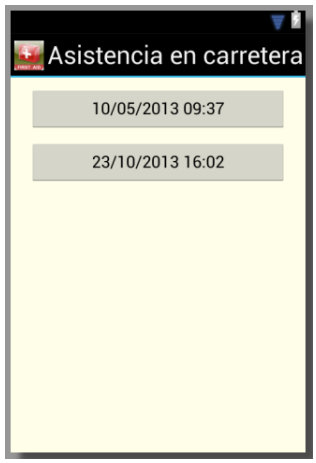


FIGURA 49: DISEÑO FINAL: LISTA DE ACCIDENTES GUARDADOS

En este caso, únicamente tenemos la sucesión de *Button* correspondientes a los distintos accidentes.

4.4. IMPLEMENTACIÓN DEL SOFTWARE

➢ Estructura de la implementación

La aplicación puede dividirse en varias partes que se comunican entre sí para completar las funcionalidades requeridas.

La primera de estas partes la constituyen las distintas *Activity* que van a definir las acciones de cada pantalla del flujo. Para simplificar su implementación, se ha diseñado una clase padre que implementa las funcionalidades comunes, extendiendo el resto de esta para incluir sus características propias.

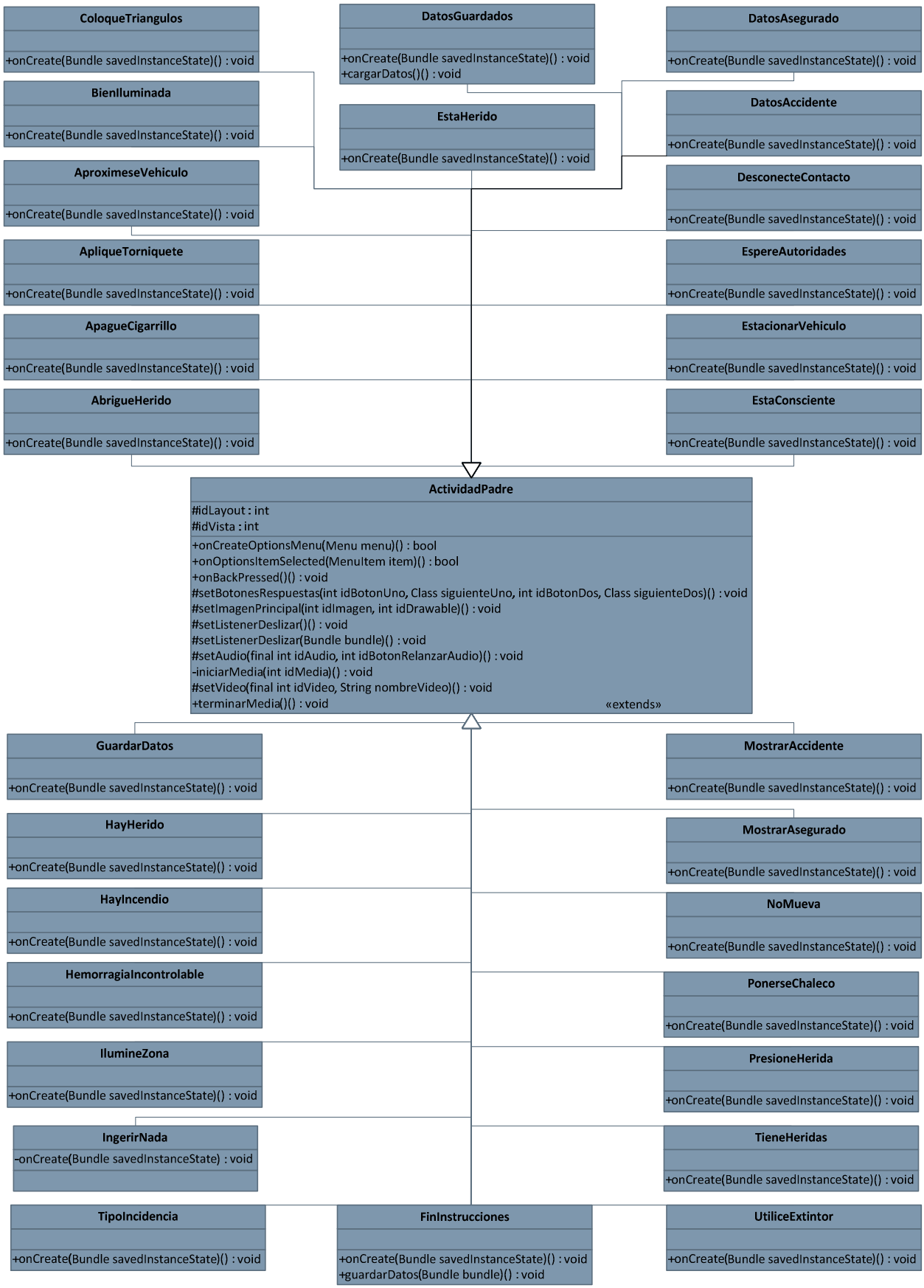


FIGURA 50: DIAGRAMA DE CLASES I: ACTIVIDADPADRE Y ACTIVIDADES QUE HEREDAN DE ELLA

Fuera de estas extensiones de la clase padre se quedan las clases *AviseAutoridades* y *VerFotos*. La primera porque requiere extender de una clase específica para poder cargar los mapas de Google y la segunda porque sirve únicamente de visor para las fotos hechas previamente de un accidente, y por lo tanto, no necesita las funcionalidades comunes de las otras actividades.

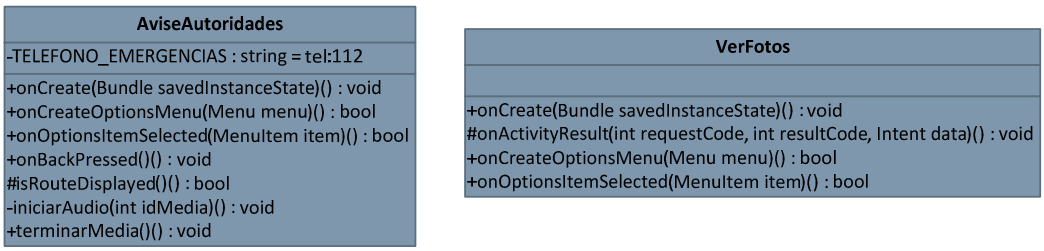


FIGURA 51: DIAGRAMA DE CLASES II: AVISEAUTORIDADES Y VERFOTOS

La segunda parte de la implementación la componen las clases que se encargan de proveer de funcionalidades comunes a modo de utilidades. Estas clases son utilizadas mayoritariamente por *ActividadPadre* y *AviseAutoridades*, aunque hay otras actividades que por características especiales requieren utilizar alguna de las utilidades. Incluso el uso es extensible entre las propias utilidades.

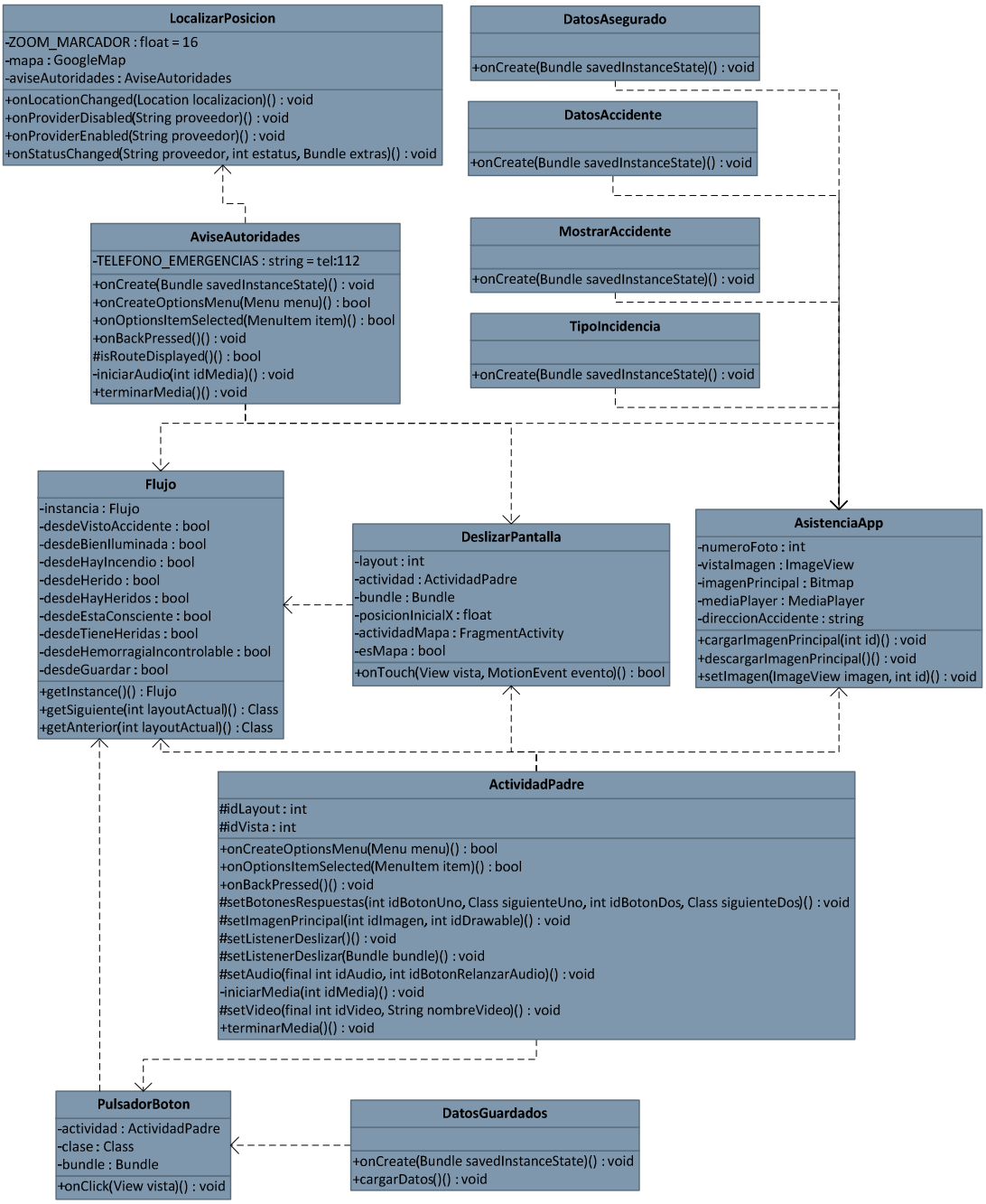


FIGURA 52: DIAGRAMA DE CLASES III: UTILIDADES

Finalmente tenemos la comunicación con la base de datos, que consta de tres tipos de comunicaciones: creación y acceso a la base de datos, insertar registros en la base de datos y leer registros. Cada funcionalidad va a ser desempeñada por una clase distintas. De esta forma, la clase *AccesoSQLite* se encargará de la creación de la base de datos y de gestionar su acceso, la clase *FinInstrucciones* será la que realice las inserciones

siempre y cuando el usuario haya introducido datos y la clase *DatosGuardados* realizará la lectura de información almacenada.

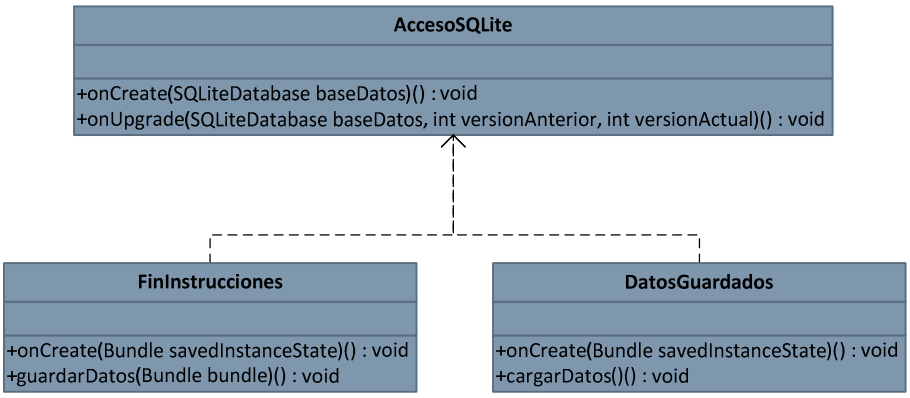


FIGURA 53: DIAGRAMA DE CLASES IV: ACCESO A LA BASE DE DATOS

- Utilidades para optimizar el software

Como hemos visto en la estructura de la implementación, tenemos una serie de clases de utilidades que son utilizadas por el resto de clases. A la hora de realizar la implementación de las distintas funcionalidades, hay que tener en cuenta que muchas de ellas tienen elementos comunes que pueden extraerse de manera que se pueda conseguir un software más optimizado. La gran mayoría de las utilidades se han diseñado para obtener este objetivo.

En primer lugar se ha decidido extraer la información del flujo que deben llevar las distintas pantallas, de manera que este esté disponible fácilmente desde cualquier punto de la aplicación. Para asegurar que esta clase es única y que siempre se va a acceder a la misma instancia desde cualquier punto de la aplicación, se utiliza el patrón de diseño *Singleton*. Para ello definimos un atributo que represente la instancia del objeto junto con un método público para obtener dicha instancia. Si es la primera vez que se intenta acceder a la instancia, se llamará a un constructor privado que la generará, mientras que los futuros accesos devolverán la instancia ya creada.

```
public class Flujo {

  private static Flujo instancia;
  // Otros atributos

  private Flujo() {
    // Código del constructor
  }

  public static Flujo getInstance() {
    if (instancia == null) {
      instancia = new Flujo();
    }
    return instancia;
  }

  // Resto de métodos
}
```

FIGURA 54: DEFINICIÓN DE LA CLASE FLUJO

Dado que todas las pantallas se cargan a través de una *Activity* que contiene el identificador de la vista correspondiente, se ha optado por establecer el flujo en función de estos identificadores de las vistas. El funcionamiento es sencillo, a partir del identificador de la vista actual, se tendrán dos métodos que devuelven cuál es la *Activity* (que es una clase Java) correspondiente a la vista anterior y cuál es la correspondiente a la vista siguiente.

```
public Class getSiguiente(int layoutActual) {
  Class siguiente = null;

  switch (layoutActual) {
    case R.layout.estacionar_vehiculo:
      siguiente = PonerseChaleco.class;
    case R.layout.ponerse_chaleco:
      siguiente = ColoqueTriangulos.class;
    case R.layout.coloque_triangulos:
      if (this.isDesdeVistoAccidente()) {
        siguiente = ApagueCigarrillo.class;
      } else {
        siguiente = AviseAutoridades.class;
      }
    case R.layout.apague_cigarrillo:
      siguiente = BienIlluminada.class;
    // Resto de casos
  }

  return siguiente;
}
```

```
public Class getAnterior(int layoutActual) {
  Class anterior = null;

  switch (layoutActual) {
    case R.layout.estacionar_vehiculo:
      if (this.isDesdeVistoAccidente()) {
        anterior = TipoIncidencia.class;
      } else {
        anterior = EstaHerido.class;
      }
    case R.layout.ponerse_chaleco:
      anterior = EstacionarVehiculo.class;
    case R.layout.coloque_triangulos:
      anterior = PonerseChaleco.class;
    case R.layout.apague_cigarrillo:
      anterior = ColoqueTriangulos.class;
    // Resto de casos
  }

  return anterior;
}
```

FIGURA 55: MÉTODOS GETSIGUIENTE Y GETANTERIOR DE LA CLASE FLUJO

Como puede verse en los métodos, hay casos en los que una misma pantalla puede llevar a varias, en función del camino que esté siguiendo el usuario desde el principio. Para controlar estos casos, se definen una serie de atributos booleanos, que permitirán saber cuál es la pantalla anterior o siguiente, según las elecciones que ha hecho previamente el usuario para llegar hasta ese punto. Por ejemplo, el atributo *desdeVistoAccidente* se fijará a *true* si el usuario selecciona la respuesta “He visto un

accidente" en la pantalla inicial y a *false* si selecciona la respuesta "He tenido un accidente".

Una vez que tenemos definido el flujo, vemos que para moverse por él hay dos maneras principales: pulsando un botón de respuesta a una pregunta o deslizando el dedo en un sentido u otro. Dado que el comportamiento de estas dos acciones es el mismo en todas las pantallas, conviene extraer las implementaciones a clases genéricas que sean utilizadas por las *Activity* sin necesidad de repetir el código en todas ellas.

En el caso de pulsar un botón, estos deben implementar la funcionalidad definida en la interfaz *OnClickListener* definida por el SDK de Android. Esta interfaz contiene un método *onClick* que deberá implementarse con el comportamiento que nosotros requerimos. Para que la clase funcione correctamente, incluimos dos constructores que nos permitirán fijar los atributos. Estos consisten en un objeto correspondiente a la *Activity* asociada a la pantalla en la que está el botón, un objeto correspondiente a la *Activity* a la cual hay que ir al pulsarse el botón y los datos a transmitir entre las *Activity*, en caso de que los haya.

```
public class PulsadorBoton implements OnClickListener {

  private ActividadPadre actividad;
  private Class clase;
  private Bundle bundle;

  public PulsadorBoton(ActividadPadre actividad, Class clase) {
    this(actividad, clase, null);
  }

  public PulsadorBoton(ActividadPadre actividad, Class clase, Bundle bundle) {
    this.actividad = actividad;
    this.clase = clase;
    this.bundle = bundle;
  }

  // Métodos
}
```

FIGURA 56: DEFINICIÓN DE LA CLASE PULSADORBOTÓN

Nuestra clase *PulsadorBoton* tendrá únicamente el método que es necesario sobreescribir. Este método recibirá el elemento correspondiente al botón pulsado (objeto de tipo *View)*, y en función de él, fijará alguno de los atributos booleanos de la clase *Flujo*, si es necesario indicar un camino concreto de cara al futuro y lanzará la *Activity* correspondiente.

```
public void onClick(View vista) {
  this.actividad.terminarMedia();

  if(clase == null) {
    // Si se ha pulsado salir, salimos del programa
    Intent startMain = new Intent(Intent.ACTION_MAIN);
    startMain.addCategory(Intent.CATEGORY_HOME);
    startMain.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
    vista.getContext().startActivity(startMain);
  } else {
    // En caso contrario mostramos la vista correspondiente al boton pulsado

    // Si pulsamos algún botón que marca un camino concreto, lo seteamos
    if (vista.getId() == R.id.botonVistoAccidente) {
      Flujo.getInstance().setDesdeVistoAccidente(true);
    } else if (vista.getId() == R.id.botonTenidoAccidente) {
      Flujo.getInstance().setDesdeVistoAccidente(false);
    } else if // Resto de condiciones especiales

    // Creamos el Intent y añadimos el bundle si existe
    Intent intent = new Intent(vista.getContext(), this.clase);
    if(bundle != null) {
      intent.putExtras(bundle);
    }

    // Iniciamos la nueva actividad
    vista.getContext().startActivity(intent);
    // Añadimos la animacion
    this.actividad.overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
  }
}
```

FIGURA 57: MÉTODO ONCLICK DE LA CLASE PULSADORBOTON

El caso para controlar el deslizado es similar al anterior, pero hay que tener algunas cosas más en cuenta. En este caso, la interfaz definida en el SDK de Android que debemos implementar es *OnTouchListener*, que en este caso hace necesario implementar un método *onTouch*. Los constructores que necesitaremos incluyen la vista actual en la que estamos (a través del identificador de su *layout*), la *Activity* en la que estamos, que puede ser una genérica o la correspondiente al mapa, un booleano para saber si estamos en una genérica o en el mapa y los datos a transmitir entre las *Activity* en caso de que los haya.

```
public class DeslizarPantalla implements OnTouchListener {

  private int layout;
  private ActividadPadre actividad;
  private Bundle bundle;
  private float posicionInicialX;
  private FragmentActivity actividadMapa;
  private boolean esMapa;

  public DeslizarPantalla(int layout, FragmentActivity actividad) {
    this.layout = layout;
    this.actividadMapa = actividad;
    this.bundle = null;
    this.esMapa = true;
  }

  public DeslizarPantalla(int layout, ActividadPadre actividad) {
    this(layout, actividad, null);
  }

  public DeslizarPantalla(int layout, ActividadPadre actividad, Bundle bundle) {
    this.layout = layout;
    this.actividad = actividad;
    this.bundle = bundle;
    this.esMapa = false;
  }

  // Métodos
}
```

FIGURA 58: DEFINICIÓN DE LA CLASE DESLIZARPANTALLA

Como puede verse en el código, se incluye un atributo *posicionInicialX*. Este atributo es utilizado por el método *onTouch* para almacenar el píxel en el cual se ha tocado la pantalla. A partir de él, cuando el usuario deja de tocar la pantalla, calculamos la distancia en píxeles recorrida por el dedo para saber si debemos avanzar, retroceder o no hacer nada en caso de que no se haya desplazado un mínimo. En caso de que haya que cambiar de pantalla, a partir del *layout* definido como atributo, accedemos a la instancia de *Flujo* y obtenemos que *Activity* es la que debemos cargar, añadiendo los datos a transmitir si los hay.

```
public boolean onTouch(View vista, MotionEvent evento) {
  switch (evento.getAction()) {
    case MotionEvent.ACTION_DOWN: // Cuando el usuario toca la pantalla por primera vez
      posicionInicialX = evento.getX();
      return true;
    case MotionEvent.ACTION_UP: // Cuando el usuario deja de presionar
      float distance = posicionInicialX - evento.getX();
      // Dejamos unos margenes para no deslizar si se pulsa la pantalla
      if (distance < -10) {
        if(!this.esMapa) {
          this.actividad.terminarMedia();
        }
        // Hemos movido de izquierda a derecha, así que mostramos el anterior
        Flujo flujo = Flujo.getInstance();
        Class actividadAnterior = flujo.getAnterior(this.layout);
        if (actividadAnterior != null) {
          // Creamos el Intent
          Intent intent = new Intent(vista.getContext(), actividadAnterior);
          // Si tenemos datos, los añadimos (necesario para volver a las pantallas de datos)
          if(this.bundle != null) {
            intent.putExtras(this.bundle);
          }
          // Iniciamos la nueva actividad
          vista.getContext().startActivity(intent);
          // Añadimos la animacion
          if(this.esMapa) {
            this.actividadMapa.overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);
          } else {
            this.actividad.overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);
          }
        }
      } else if (distance > 10) {
        if(!this.esMapa) {
          this.actividad.terminarMedia();
        }
        // Hemos movido de derecha a izquierda, así que mostramos el siguiente
        Flujo flujo = Flujo.getInstance();
        Class actividadSiguiente = flujo.getSiguiente(this.layout);
        if (actividadSiguiente != null) {
          // Creamos el Intent
          Intent intent = new Intent(vista.getContext(), actividadSiguiente);
          // Si tenemos datos, los añadimos (necesario para volver a las pantallas de datos)
          if(this.bundle != null) {
            intent.putExtras(this.bundle);
          }
          // Iniciamos la nueva actividad
          vista.getContext().startActivity(intent);
          // Añadimos la animacion
          if(this.esMapa) {
            this.actividadMapa.overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
          } else {
            this.actividad.overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
          }
        }
      }
    default:
      break;
  }
  return false;
}
```

FIGURA 59: MÉTODO ONTOUCH DE LA CLASE DESLIZARPANTALLA

Finalmente, tenemos una última clase que nos permite optimizar la memoria que utiliza la aplicación mientras se está ejecutando. Dado que hay que cargar varios ficheros de imágenes y varios de reproducción (ya sean audios o vídeos), conviene prestar especial atención a que no se queden esos ficheros ocupando memoria innecesariamente. Para evitarlo, se ha creado una clase que hereda de *Application*. Esta va a ser la clase "general" que mantenga el estado de toda la aplicación, y que será visible a cualquier otra clase. Para ello basta con indicar esta clase en el archivo *AndroidManifest.xml* mediante el tag <*application*>. Cada vez que se quiera cargar una

nueva foto o un nuevo archivo de reproducción, estos deberán ser fijados en esta clase, ya que se utilizará el último de ellos que esté fijado. Esto nos permitirá mantener en memoria como mucho una imagen y un archivo de reproducción únicamente. Además de esto, la clase contará con un contador que indicará el número de fotos realizadas en caso de tener un accidente, lo que nos permitirá tener un nombre secuencial con el que identificar las fotos de un mismo accidente y con la ubicación en la cual se ha producido el accidente.

```
public class AsistenciaApp extends Application {

  private int numeroFoto;
  private ImageView vistaImagen;
  private Bitmap imagenPrincipal;
  private MediaPlayer mediaPlayer;
  private String direccionAccidente;

  public void cargarImagenPrincipal(int id) {
    imagenPrincipal = BitmapFactory.decodeStream(getResources().openRawResource(id));
    vistaImagen.setImageBitmap(imagenPrincipal);
  }

  public void descargarImagenPrincipal() {
    if (vistaImagen != null) {
      vistaImagen.setImageBitmap(null);
    }
    if (imagenPrincipal != null) {
      imagenPrincipal.recycle();
    }
    imagenPrincipal = null;
  }

  public void setImagen(ImageView imagen, int id) {
    descargarImagenPrincipal();
    vistaImagen = imagen;
    cargarImagenPrincipal(id);
  }

  // Getters y setters
}
```

FIGURA 60: DEFINICIÓN DE LA CLASE ASISTENCIAAPP

- Actividades del flujo

La parte principal de la implementación engloba las distintas *Activity* que forman el flujo de pantallas de la aplicación. Como se vio en la estructura de la implementación, la gran mayoría de las actividades heredan de una actividad padre común denominada *ActividadPadre*. Para poder saber qué actividad hija está ejecutando en cada momento, la actividad padre define dos atributos: *idVista* e *idLayout*. Estos atributos representan el identificador de la vista correspondiente a la *Activity* y el identificador de su pantalla *xml*.

Dentro de los métodos comunes que implementa podemos hacer tres grupos. El primero de ellos corresponde con la implementación de los métodos necesarios de la

clase *Activity* definida en el SDK de Android. Estos son *onCreateOptionsMenu* para crear el menú común, *onOptionsItemSelected* para capturar cuando se seleccione una de las opciones del menú y *onBackPressed* que captura cuando se pulsa el botón de volver del dispositivo.

```
public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.menu, menu);
  return true;
}

public boolean onOptionsItemSelected(MenuItem item) {
  final View vista = (View) findViewById(idVista);
  Intent intent;
  this.terminarMedia();
  switch (item.getItemId()) {
    case R.id.menuSOS:
      intent = new Intent(this, TipoIncidencia.class);
      vista.getContext().startActivity(intent);
      this.overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
      break;
    case R.id.menuDatosGuardados:
      intent = new Intent(this, DatosGuardados.class);
      vista.getContext().startActivity(intent);
      this.overridePendingTransition(R.anim.push_left_in, R.anim.push_left_out);
      break;
    case R.id.menuAyuda:
      break;
    default:
      break;
  }
  return true;
}

public void onBackPressed() {
  this.terminarMedia();

  // Simulamos que hemos deslizado hacia atrás
  final View vista = (View) findViewById(idVista);

  Flujo flujo = Flujo.getInstance();
  Class actividadAnterior = flujo.getAnterior(idLayout);
  Bundle bundle = this.getIntent().getExtras();

  if (actividadAnterior != null) {
    // Creamos el Intent
    Intent intent = new Intent(vista.getContext(), actividadAnterior);

    // Si tenemos datos, los añadimos (necesario para volver a las
    // pantallas de datos)
    if (bundle != null) {
      intent.putExtras(bundle);
    }

    // Iniciamos la nueva actividad
    vista.getContext().startActivity(intent);

    // Añadimos la animacion
    this.overridePendingTransition(R.anim.push_right_in, R.anim.push_right_out);
  }
}
```

FIGURA 61: MÉTODOS ONCREATEOPTIONSMENU, ONOPTIONSITEMSELECTED Y ONBACKPRESSED DE ACTIVIDADPADRE

El segundo de los grupos corresponde a las funciones que se encargan del comportamiento de la pantalla. Es decir, aquellas que controlan que imagen se va a mostrar (recordemos que se implementó una optimización por la cual solo se puede

mantener una imagen en memoria), el comportamiento de los botones de respuesta y el comportamiento en caso de deslizar el dedo por la pantalla.

```
protected void setBotonesRespuestas(int idBotonUno, Class siguienteUno, int idBotonDos,
    Class siguienteDos) {
  final Button botonUno = (Button) findViewById(idBotonUno);
  final Button botonDos = (Button) findViewById(idBotonDos);

  PulsadorBoton pulsadoBotonUno = new PulsadorBoton(this, siguienteUno);
  botonUno.setOnClickListener(pulsadoBotonUno);

  PulsadorBoton pulsadoBotonDos = new PulsadorBoton(this, siguienteDos);
  botonDos.setOnClickListener(pulsadoBotonDos);
}

protected void setImagenPrincipal(int idImagen, int idDrawable) {
  AsistenciaApp imagenesCargadas = (AsistenciaApp) getApplication();
  imagenesCargadas.setImagen((ImageView) findViewById(idImagen), idDrawable);
}

protected void setListenerDeslizar() {
  this.setListenerDeslizar(null);
}

protected void setListenerDeslizar(Bundle bundle) {
  // Añadimos el listener para el evento onTouch
  final View vista = (View) findViewById(idVista);
  DeslizarPantalla deslizarPantalla = new DeslizarPantalla(idLayout, this, bundle);
  vista.setOnTouchListener(deslizarPantalla);
}
```

FIGURA 62: MÉTODOS SETBOTONESRESPUESTAS, SETIMAGENPRINCIPAL Y SETLISTENERDESLIZAR DE ACTIVIDADPADRE

Como puede verse, esto métodos se han implementado con visibilidad *"protected"*. Esto es debido a que son métodos que deberán ser accedidos únicamente por las clases hijas, de manera que evitamos que clases externas modifiquen el comportamiento que debe tener una actividad.

El último grupo de métodos se corresponde con los ficheros de reproducción que se van a incluir en cada actividad. Estos ficheros incluyen audios y videos, y al igual que con las imágenes, solo puede haber un archivo de reproducción en cada momento en la memoria de la aplicación, por lo que es necesario fijarlos al principio, y liberarlos una vez sean utilizados.

```
protected void setAudio(final int idAudio, int idBotonRelanzarAudio) {
  final ImageView botonRelanzarAudio = (ImageView) findViewById(idBotonRelanzarAudio);
  this.iniciarMedia(idAudio);

  // Añadimos el control para los botones del volumen y lanzamos el audio al cargar la página
  setVolumeControlStream(AudioManager.STREAM_MUSIC);
  ((AsistenciaApp) getApplication()).getMediaPlayer().start();

  // Se controla el evento para liberar el audio una vez terminado
  ((AsistenciaApp) getApplication()).getMediaPlayer().setOnCompletionListener(new OnCompletionListener() {
          public void onCompletion(MediaPlayer mediaPlayer) {
            mediaPlayer.release();
          }

      });

  // Si se pulsa el boton de relanzar, se para y se comienza de nuevo
  if(botonRelanzarAudio != null) {
    botonRelanzarAudio.setOnClickListener(new OnClickListener() {
            public void onClick(View view) {
              iniciarMedia(idAudio);
              ((AsistenciaApp) getApplication()).getMediaPlayer().start();
            }

        });
  }
}

private void iniciarMedia(int idMedia) {
  if(((AsistenciaApp) getApplication()).getMediaPlayer() != null) {
    this.terminarMedia();
  }
  ((AsistenciaApp) getApplication()).setMediaPlayer(MediaPlayer.create(ActividadPadre.this, idMedia));
}

protected void setVideo(final int idVideo, String nombreVideo) {
  final VideoView videoView = (VideoView) findViewById(idVideo);
  Uri videoPath = Uri.parse("android.resource://es.uc3m.pfc.asistenciaencarretera/raw/" + nombreVideo);
  videoView.setVideoURI(videoPath);
  videoView.setMediaController(new MediaController(this));
  videoView.requestFocus();
  videoView.start();
}

public void terminarMedia() {
  if(((AsistenciaApp) getApplication()).getMediaPlayer() != null) {
    try {
      ((AsistenciaApp) getApplication()).getMediaPlayer().stop();
      ((AsistenciaApp) getApplication()).getMediaPlayer().release();
    } catch (Exception e) {
      // Si no habia media inicializado
    }
  }
}
```

FIGURA 63: MÉTODOS SETAUDIO, INICIARMEDIA, SETVIDEO Y TERMINARMEDIA DE ACTIVIDADPADRE

Una vez definidas las funcionalidades comunes, las distintas actividades hijas quedan muy sencillas, de manera que lo único que tenemos que hacer es fijar los atributos definidos en la clase padre y llamar a los métodos oportunos para cargar las imágenes, botones y/o ficheros de reproducción necesarios. A continuación vemos un ejemplo de cómo quedaría una clase con imagen y otra con botones:

```java
public class EstaHerido extends ActividadPadre {

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.esta_herido);
    this.idLayout = R.layout.esta_herido;
    this.idVista = R.id.vistaEstaHerido;

    this.setBotonesRespuestas(R.id.botonSiHerido,
      AviseAutoridades.class, R.id.botonNoHerido,
      EstacionarVehiculo.class);
    this.setListenerDeslizar();
    this.setAudio(R.raw.preg_herido2, -1);
  }
}
```

```java
public class EstacionarVehiculo extends ActividadPadre {

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.estacionar_vehiculo);
    this.idLayout = R.layout.estacionar_vehiculo;
    this.idVista = R.id.vistaEstacionarVehiculo;

    this.setImagenPrincipal(R.id.estacionarImg,
      R.drawable.estacionar_img);
    this.setListenerDeslizar();
    this.setAudio(R.raw.aparcar, R.id.estacionarVehiculoAudio);
  }
}
```

FIGURA 64: EJEMPLO DE DEFINICIÓN DE ACTIVIDADES HIJAS DE ACTIVIDADPADRE

Únicamente hay dos actividades que se salen de esta norma de definirse de manera sencilla debido a que no heredan de la funcionalidad básica de la actividad padre. Estas actividades son *AviseAutoridades* y *VerFotos*.

En el caso de *AviseAutoridades*, al incluir un mapa de Google Maps, es necesario que extienda de una clase específica, *FragmentActivity*. Dado que va a mantener varias funcionalidades ya definidas en la actividad padre, lo único que se ha hecho ha sido copiar dichas funcionalidades en la clase. Estas incluyen *onCreateOptionsMenu*, *onOptionsItemSelected*, *onBackPressed* e *iniciarAudio*. De esta forma, la funcionalidad que difiere se centra en la creación y la inclusión de un método necesario para el mapa, *isRouteDisplayed*, que devolverá siempre falso ya que no se van a mostrar rutas.

```
public class AviseAutoridades extends FragmentActivity {

  private static final String TELEFONO_EMERGENCIAS = "tel:112";

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.avise_autoridades);
    this.iniciarAudio(R.raw.mapa);

    // Implementamos la llamada si se pulsa el boton
    final Button llamada = (Button) findViewById(R.id.botonLlamada);
    llamada.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse(TELEFONO_EMERGENCIAS));
        startActivity(intent);
      }
    });

    // Obtenemos la referencia al mapa y fijamos el tipo de mapa que
    // mostraremos
    GoogleMap mapa = ((SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.mapaLocalizacion))
        .getMap();
    mapa.setMapType(GoogleMap.MAP_TYPE_HYBRID);

    // Obtenemos una referencia al LocationManager
    LocationManager locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

    // Nos registramos para recibir actualizaciones de la posición del GPS y
    // por la red
    LocalizarPosicion localizarPosicion = new LocalizarPosicion(mapa);
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, localizarPosicion);
    locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, localizarPosicion);

    // Añadimos el listener para el evento onTouch
    final View vista = (View) findViewById(R.id.vistaAviseAutoridades);
    DeslizarPantalla deslizarPantalla = new DeslizarPantalla(R.layout.avise_autoridades, this);
    vista.setOnTouchListener(deslizarPantalla);
  }

  protected boolean isRouteDisplayed() {
    // Como no queremos mostrar rutas en el mapa, se devuelve false
    return false;
  }

  //Resto de métodos copiados de ActividadPadre
}
```

FIGURA 65: DEFINICIÓN DE AVISEAUTORIDADES

Como puede verse, en la creación, aparte de cargar el mapa y obtener la posición en la que nos encontramos, también se incluye la funcionalidad para llamar a los servicios de emergencias en caso de pulsar el botón correspondiente. Además, se hace uso de la clase *LocalizarPosicion*. Esta es una clase sencilla que implementa *LocationListener* del SDK de Android y simplemente implementa su método *onLocationChanged*, dejando los demás métodos sin implementar. Aparte de eso, únicamente fija un zoom inicial con el que mostrar el mapa.

```
public class LocalizarPosicion implements LocationListener {

  private static final float ZOOM_MARCADOR = 16;
  private GoogleMap mapa;
  private AviseAutoridades aviseAutoridades;

  public LocalizarPosicion(GoogleMap mapa, AviseAutoridades aviseAutoridades) {
    this.mapa = mapa;
    this.aviseAutoridades = aviseAutoridades;
  }

  public void onLocationChanged(Location localizacion) {
    // Añadimos el marcador en la posición actual y centramos el mapa en el
    LatLng posicion = new LatLng(localizacion.getLatitude(), localizacion.getLongitude());
    mapa.clear();

    // Obtenemos la direccion y la almacenamos para usarla al guardar los datos del accidente
    Geocoder gc = new Geocoder(aviseAutoridades, Locale.getDefault());
    StringBuilder direccion = new StringBuilder();
        try {
      List<Address> addresses = gc.getFromLocation(posicion.latitude, posicion.longitude, 1);
      String calle = addresses.get(0).getAddressLine(0);
      String ciudad = addresses.get(0).getAddressLine(1);
      direccion.append(calle);
      direccion.append("(");
      direccion.append(ciudad);
      direccion.append(")");
    } catch (Exception e) {
      direccion.append("No se ha encontrado la dirección.");
    }
        ((AsistenciaApp) aviseAutoridades.getApplication()).setDireccionAccidente(direccion.toString());

    mapa.addMarker(new MarkerOptions().position(posicion).title(direccion.toString()));
    mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(posicion, ZOOM_MARCADOR));
  }

  public void onProviderDisabled(String proveedor) {
    // No hacemos nada
  }

  public void onProviderEnabled(String proveedor) {
    // No hacemos nada
  }

  public void onStatusChanged(String proveedor, int estatus, Bundle extras) {
    // No hacemos nada
  }
}
```

FIGURA 66: DEFINICIÓN DE LOCALIZARPOSICION

Como puede verse, esta clase contiene una referencia a *AviseAutoridades*. Esta referencia se utilizará para poder acceder a la clase *AsistenciaApp* que contiene las características globales de la aplicación. En dichas características se almacenará la ubicación del accidente una vez obtenida, para que esté disponible para futuros usos.

El caso de *VerFotos* es muy sencillo, ya que únicamente tendremos un botón para cargar una foto y verla ampliada. De este modo, únicamente tendremos que incluir el evento para manejar el botón que se encargue de cargar fotos y una vez seleccionada la foto, mostrarla en la pantalla mediante el método *onActivityResult*.

```
public class VerFotos extends Activity {

  private static int RESULT_LOAD_IMAGE = 1;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.ver_fotos);

        Button botonVolver = (Button) findViewById(R.id.botonVolver);
        botonVolver.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View vista) {
                Intent i = new Intent(
                        Intent.ACTION_PICK,
                        android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
                startActivityForResult(i, RESULT_LOAD_IMAGE);
            }
        });
    }

    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == RESULT_LOAD_IMAGE && resultCode == RESULT_OK && null != data) {
            Uri selectedImage = data.getData();
            String[] filePathColumn = { MediaStore.Images.Media.DATA };

            Cursor cursor = getContentResolver().query(selectedImage,
                    filePathColumn, null, null, null);
            cursor.moveToFirst();

            int columnIndex = cursor.getColumnIndex(filePathColumn[0]);
            String picturePath = cursor.getString(columnIndex);
            cursor.close();

            ImageView verImagen = (ImageView) findViewById(R.id.verImagen);
            verImagen.setImageBitmap(BitmapFactory.decodeFile(picturePath));
        }
    }
}
```

FIGURA 67: DEFINICIÓN DE VERFOTOS

Podemos destacar las actividades correspondientes a la recogida de datos de un accidente y del asegurado. Estas actividades, además de implementar la actividad padre, deben encargarse de recoger información que posteriormente será almacenada en la base de datos.

En primer lugar tenemos los datos del accidente. Esta actividad vamos a dividirla en varios grupos de acciones. Por un lado, tenemos la inicialización de la actividad. Esta se lleva a cabo de igual manera que el resto de actividades, incluyendo un archivo de audio únicamente. Posteriormente, tendremos la definición de los elementos correspondientes a los distintos campos de texto donde el usuario podrá rellenar la información. En caso de que tengamos ya información de esos campos (por ejemplo, si el usuario ha retrocedido a este paso desde el siguiente), se rellenarán los campos con dicha información. En el caso de la fecha y el lugar, en caso de no tener la información, se rellenará automáticamente con la fecha actual y con la posición obtenida desde el mapa.

```
public class DatosAccidente extends ActividadPadre {
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.datos_accidente);
    this.idLayout = R.layout.datos_accidente;
    this.idVista = R.id.vistaDatosAccidente;

    // Vemos si tenemos información pasada en el intent (por si volvemos
    // desde DatosAsegurado)
    Bundle bundle = this.getIntent().getExtras();

    this.setListenerDeslizar(bundle);
    this.setAudio(R.raw.datos, -1);

    // Obtenemos una referencia a los controles de la interfaz
    final EditText textoFechaHora = (EditText) findViewById(R.id.fechaHora);
    final EditText textoLugar = (EditText) findViewById(R.id.lugar);
    final EditText textoDescripcion = (EditText) findViewById(R.id.descripcion);
    final CheckBox checkContrario = (CheckBox) findViewById(R.id.contrario);

    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm");
    String fechaHora = sdf.format(Calendar.getInstance().getTime());
    textoFechaHora.setText(fechaHora);
    textoLugar.setText(((AsistenciaApp) getApplication()).getDireccionAccidente());

    if (bundle != null) {
      // Comprobamos la información del bundle por si viene desde
      // DatosAsegurado ponerla
      if (bundle.get("fechaHora") != null && ((String) bundle.get("fechaHora")).length() > 0) {
        textoFechaHora.setText((String) bundle.get("fechaHora"));
      }
      if (bundle.get("lugar") != null && ((String) bundle.get("lugar")).length() > 0) {
        textoLugar.setText((String) bundle.get("lugar"));
      }
      if (bundle.get("descripcion") != null && ((String) bundle.get("descripcion")).length() > 0) {
        textoDescripcion.setText((String) bundle.get("descripcion"));
      }
    }

    // Implementación de los eventos de los botones
  }
}
```

FIGURA 68: DEFINICIÓN DE DATOSACCIDENTE

Por último, se implementará el comportamiento al pulsar el botón para tomar fotos y el correspondiente al botón para guardar los datos. En el primer caso, se hará uso del contador de fotos disponible en *AsistenciaApp*, para componer el nombre con ese valor y la fecha del accidente. A la hora de pulsar guardar, simplemente se recogerán los datos introducidos por el usuario (o introducidos por defecto).

```
public class DatosAccidente extends ActividadPadre {
  protected void onCreate(Bundle savedInstanceState) {
    // Creación de la actividad

    // Implementamos el evento del botón tomar fotos
    final Button tomarFoto = (Button) findViewById(R.id.botonTomarFotos);
    tomarFoto.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");

        StringBuilder pathName = new StringBuilder(Environment.getExternalStorageDirectory().getAbsolutePath());
        pathName.append(PATH_FOTOS);
        File carpeta = new File(pathName.toString());
        if (!carpeta.exists()) {
          carpeta.mkdirs();
        }
        StringBuilder nombreFoto = new StringBuilder(PATH_FOTOS);
        int numeroFoto = ((AsistenciaApp) getApplication()).getNumeroFoto();
        nombreFoto.append(textoFechaHora.getText().toString());
        nombreFoto.append("_");
        nombreFoto.append(numeroFoto);
        nombreFoto.append(".jpg");
        ((AsistenciaApp) getApplication()).setNumeroFoto(numeroFoto++);

        File foto = new File(carpeta, nombreFoto.toString());
        intent.putExtra(MediaStore.EXTRA_OUTPUT,Uri.fromFile(foto));
        startActivity(intent);
      }
    });

    // Implementamos el evento del botón guardar
    final Button guardar = (Button) findViewById(R.id.guardarDatosAccidente);
    guardar.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        // Creamos el Intent
        Intent intent = new Intent(DatosAccidente.this, DatosAsegurado.class);

        // Creamos la información a pasar entre actividades
        String fechaHora = textoFechaHora.getText().toString();
        String lugar = textoLugar.getText().toString();
        String descripcion = textoDescripcion.getText().toString();

        Bundle b = new Bundle();
        b.putString("fechaHora", fechaHora);
        b.putString("lugar", lugar);
        b.putString("descripcion", descripcion);

        intent.putExtras(b);
        startActivity(intent);
      }
    });
  }
}
```

FIGURA 69: DETALLE DE LOS EVENTOS DE DATOSACCIDENTE

En el caso de los datos del asegurado, contaremos con tres partes. La primera de ellas correspondiente a la creación de la pantalla. La segunda, corresponderá a los datos mostrados en la pantalla, que se rellenarán automáticamente en caso de que estemos volviendo a la pantalla después de que el usuario ya los haya escrito. En este caso, además de contar con la información de esta pantalla, también se obtendrá la información de los datos del accidente, ya que estos serán pasados a esta actividad.

```
public class DatosAsegurado extends ActividadPadre {
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.datos_asegurado);
    this.idLayout = R.layout.datos_asegurado;
    this.idVista = R.id.vistaDatosAsegurado;

    // Recuperamos la información pasada en el intent
    Bundle bundle = this.getIntent().getExtras();

    this.setListenerDeslizar(bundle);
    this.setAudio(R.raw.datos2, -1);

    // Obtenemos una referencia a los controles de la interfaz
    final EditText textoNombre = (EditText) findViewById(R.id.nombre);
    final EditText textoNif = (EditText) findViewById(R.id.nif);
    final EditText textoDomicilio = (EditText) findViewById(R.id.domicilio);
    final EditText textoTelefono = (EditText) findViewById(R.id.telefono);
    final EditText textoMatricula = (EditText) findViewById(R.id.matricula);
    final EditText textoMarca = (EditText) findViewById(R.id.marca);
    final EditText textoAseguradora = (EditText) findViewById(R.id.aseguradora);
    final EditText textoPoliza = (EditText) findViewById(R.id.poliza);
    final Button guardar = (Button) findViewById(R.id.guardarDatosAsegurado);

    final String fechaHora = bundle.getString("fechaHora");
    final String lugar = bundle.getString("lugar");
    final String descripcion = bundle.getString("descripcion");

    // Comprobamos la información del bundle por si viene de FinInstrucciones ponerla
    if (bundle.get("nombre") != null && ((String) bundle.get("nombre")).length() > 0) {
      textoNombre.setText((String) bundle.get("nombre"));
    }
    if (bundle.get("nif") != null && ((String) bundle.get("nif")).length() > 0) {
      textoNif.setText((String) bundle.get("nif"));
    }
    if (bundle.get("domicilio") != null && ((String) bundle.get("domicilio")).length() > 0) {
      textoDomicilio.setText((String) bundle.get("domicilio"));
    }
    if (bundle.get("telefono") != null && ((String) bundle.get("telefono")).length() > 0) {
      textoTelefono.setText((String) bundle.get("telefono"));
    }
    if (bundle.get("matricula") != null && ((String) bundle.get("matricula")).length() > 0) {
      textoMatricula.setText((String) bundle.get("matricula"));
    }
    if (bundle.get("marca") != null && ((String) bundle.get("marca")).length() > 0) {
      textoMarca.setText((String) bundle.get("marca"));
    }
    if (bundle.get("aseguradora") != null && ((String) bundle.get("aseguradora")).length() > 0) {
      textoAseguradora.setText((String) bundle.get("aseguradora"));
    }
    if (bundle.get("poliza") != null && ((String) bundle.get("poliza")).length() > 0) {
      textoPoliza.setText((String) bundle.get("poliza"));
    }

    // Implementamos el evento "click" del botón
  }
}
```

FIGURA 70: DEFINICIÓN DE DATOSASEGURADO

En esta pantalla contaremos únicamente con un botón, que nos permitirá guardar los datos. Este botón pasará a la siguiente actividad todos los datos, tanto los del accidente como los del asegurado.

```
public class DatosAsegurado extends ActividadPadre {
  protected void onCreate(Bundle savedInstanceState) {
    // Creación

    // Implementamos el evento "click" del botón
    guardar.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        // Creamos el Intent
        Intent intent = new Intent(DatosAsegurado.this, FinInstrucciones.class);

        // Creamos la información a pasar entre actividades
        String nombre = textoNombre.getText().toString();
        String nif = textoNif.getText().toString();
        String domicilio = textoDomicilio.getText().toString();
        String telefono = textoTelefono.getText().toString();
        String matricula = textoMatricula.getText().toString();
        String marca = textoMarca.getText().toString();
        String aseguradora = textoAseguradora.getText().toString();
        String poliza = textoPoliza.getText().toString();

        Bundle b = new Bundle();
        b.putString("fechaHora", fechaHora);
        b.putString("lugar", lugar);
        b.putString("descripcion", descripcion);
        b.putString("nombre", nombre);
        b.putString("nif", nif);
        b.putString("domicilio", domicilio);
        b.putString("telefono", telefono);
        b.putString("matricula", matricula);
        b.putString("marca", marca);
        b.putString("aseguradora", aseguradora);
        b.putString("poliza", poliza);

        // Añadimos la información al intent
        intent.putExtras(b);

        // Iniciamos la nueva actividad
        startActivity(intent);
      }
    });
  }
}
```

FIGURA 71: DETALLE DE EVENTO GUARDAR DE DATOSASEGURADO

Estas dos actividades serán prácticamente idénticas a las correspondientes para mostrar los datos de accidentes y asegurados almacenados previamente. La única diferencia radica en la desaparición de los botones para guardar los datos, y el cambio del botón para tomar fotos por otro para ver las fotos hechas previamente. Es decir, se eliminan las implementaciones de ambos botones "Guardar" y se modifica ligeramente la implementación del botón "Tomar fotos".

- Acceso a la base de datos

Como se vio en el diagrama de clases, el acceso a la base de datos se lleva a cabo mediante tres clases distintas.

La clase principal, *AccesoSQLite* extiende de la clase *SQLiteOpenHelper*, que se encarga de gestionar las bases de datos de *sqlite*. En nuestro caso, vamos a sobreescribir únicamente la clase encargada de la creación de la base de datos. Esta creación la haremos a partir de las tablas identificadas en el diagrama de base de datos visto en el

diseño de la aplicación, de manera que definiremos dos variables globales finales que corresponderán a las dos expresiones para la creación de las tablas.

```
public class AccesoSQLite extends SQLiteOpenHelper {

  private static final String TABLA_ACCIDENTE =
    "CREATE TABLE ACCIDENTE (fecha TEXT, lugar TEXT, descripcion TEXT)";
  private static final String TABLA_ASEGURADO =
    "CREATE TABLE ASEGURADO (fecha TEXT, nombre TEXT, nif TEXT, domicilio TEXT, telefono TEXT, matricula TEXT, marca TEXT, aseguradora TEXT, poliza TEXT)";

  public AccesoSQLite(Context contexto, String nombre, CursorFactory factory, int version) {
    super(contexto, nombre, factory, version);
  }

  public void onCreate(SQLiteDatabase baseDatos) {
    baseDatos.execSQL(TABLA_ACCIDENTE);
    baseDatos.execSQL(TABLA_ASEGURADO);
  }

  public void onUpgrade(SQLiteDatabase baseDatos, int versionAnterior, int versionActual) {
    // No hacemos nada
  }

}
```

FIGURA 72: DEFINICIÓN DE ACCESOSQLITE

Esta clase será utilizada por las otras dos para cargar la base de datos.

Para almacenar datos nuevos en la base de datos, contamos con la clase *FinInstrucciones*. Esta clase, además de ser una actividad estándar que hereda de la actividad padre, consta de un método que recoge la información suministrada por el usuario y realiza la inserción en las tablas correspondientes.

```
private void guardarDatos(Bundle bundle) {
  // Abrimos la base de datos en modo escritura
  AccesoSQLite accesoSQLite = new AccesoSQLite(this, "DBAsistenciaCarretera", null, 1);
  SQLiteDatabase baseDatos = accesoSQLite.getWritableDatabase();
  // Si hemos abierto correctamente la base de datos
  if (bundle != null && baseDatos != null) {
    // Almacenamos los datos del accidente
    StringBuilder insertAccidente = new StringBuilder(INSERT_ACCIDENTE);
    insertAccidente.append(" values ('");
    if (bundle.get("fechaHora") != null) {
      insertAccidente.append(bundle.get("fechaHora"));
    } else {
      insertAccidente.append("-");
    }
    insertAccidente.append("','");
    // Recogida del resto de datos del accidente
    insertAccidente.append("');");
    baseDatos.execSQL(insertAccidente.toString());

    // Almacenamos los datos del asegurado si lo hay
    if (bundle.get("nombre") != null) {
      StringBuilder insertAsegurado = new StringBuilder(INSERT_ASEGURADO);
      insertAsegurado.append(" values ('");
      if (bundle.get("fechaHora") != null) {
        insertAsegurado.append(bundle.get("fechaHora"));
      } else {
        insertAsegurado.append("-");
      }
      insertAccidente.append("','");
      // Recogida del resto de datos del asegurado
      baseDatos.execSQL(insertAsegurado.toString());
    }
    insertAccidente.append("');");
    baseDatos.close();
  }
}
```

FIGURA 73: DETALLE DEL MÉTODO GUARDARDATOS DE LA ACTIVIDAD FININSTRUCCIONES

Como puede verse en el método para guardar los datos, accedemos a la información recibida de las pantallas para introducirla, y abrimos una nueva conexión a partir de la clase *AccesoSQLite*. Una vez tenemos la conexión con la base de datos, montamos las sentencias de inserción y las ejecutamos, cerrando finalmente la conexión con la base de datos.

Cuando vamos a recoger los datos almacenados en la clase *DatosGuardados*, el funcionamiento es muy similar al anterior, salvo que las sentencias de base de datos en este caso serán *select*. Al igual que con *FinInstrucciones*, esta clase se crea como cualquier otra actividad que hereda de la actividad padre, y únicamente se añade un método que se encarga de cargar los datos almacenados.

```
private void cargarDatos() {
  // Abrimos la base de datos en modo escritura
  AccesoSQLite accesoSQLite = new AccesoSQLite(this, "DBAsistenciaCarretera", null, 1);
  SQLiteDatabase baseDatos = accesoSQLite.getWritableDatabase();

  // Si hemos abierto correctamente la base de datos
  if (baseDatos != null) {
    // Leemos los datos de los accidentes
    Cursor datosAccidentes = baseDatos.rawQuery(
        "SELECT fecha, lugar, descripcion FROM ACCIDENTE", null);

    if (datosAccidentes.moveToFirst()) {
      // Recorremos el cursor hasta que no haya más registros
      do {
        String [] args = {datosAccidentes.getString(0)};
        Cursor datosAsegurado = baseDatos.rawQuery(
            "SELECT fecha, nombre, nif, domicilio, telefono, matricula, marca, aseguradora, poliza FROM ASEGURADO where fecha = ?", args);

        Bundle bundle = new Bundle();
        bundle.putString("fechaHora", datosAccidentes.getString(0));
        // Extraemos el resto de datos del accidente
        if(datosAsegurado.moveToFirst()) {
          bundle.putString("nombre", datosAsegurado.getString(1));
          // Extraemos el resto de datos del asegurado
        }

        // Añadimos un botón a la vista por cada accidente
        Button boton = new Button(this, null);
        boton.setText(datosAccidentes.getString(0));
        boton.setTextColor(Color.BLACK);
        boton.setBackgroundColor(Color.GRAY);
        LinearLayout layout = (LinearLayout) findViewById(R.id.vistaDatosGuardados);
        LayoutParams layoutParams = new LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT);
        layoutParams.setMargins(20, 10, 20, 0);
        layout.addView(boton, layoutParams);

        PulsadorBoton pulsadoBoton = new PulsadorBoton(this, MostrarAccidente.class, bundle);
        boton.setOnClickListener(pulsadoBoton);
      } while (datosAccidentes.moveToNext());
    }

    baseDatos.close();
  }
}
```

FIGURA 74: DETALLE DEL MÉTODO CARGARDATOS DE LA ACTIVIDAD DATOSGUARDADOS

Como puede verse en el método, en primer lugar cargamos todos los datos relativos a accidentes. Una vez que los tenemos, recorremos todos ellos, y por cada uno, extraemos la información del accidente de la consulta y buscamos los datos de asegurado correspondientes. En caso de que tengamos datos de asegurado, extraemos su información. Una vez tenemos toda la información de un accidente, se crea un nuevo

botón en la pantalla para que al seleccionar el accidente, se muestre toda su información.

Los cambios comentados en la información a recoger del accidente y del asegurado, suponen un cambio en el diseño inicial planteado de la base de datos. En lugar de tener una única tabla, el diseño pasa a tener dos, una correspondiente a los datos de los accidentes y otra correspondiente a los datos de los asegurados. De esta forma, podemos tener datos de accidentes en los cuales no haya contrario, y otros en los que si los haya. Ambas tablas se relacionarán a partir de la fecha del accidente, por lo que el diseño final de la base de datos se corresponderá con:

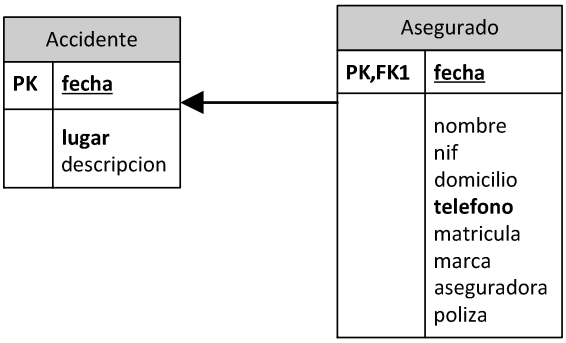


FIGURA 75: DIAGRAMA DE LA BASE DE DATOS

4.5. DIAGRAMAS DE SECUENCIA

Para mostrar la funcionalidad básica de la aplicación, vamos a analizar una serie de diagramas de secuencia que ayudarán a entender los pasos por los que pasa la aplicación para realizar las acciones principales.

En primer lugar vamos a centrarnos en el flujo correspondiente a la creación de una actividad de tipo pregunta y las acciones que esta lleva a cabo hasta que el usuario pulsa una de las opciones de respuesta. En primer lugar se realiza la inicialización de la actividad, es decir, la ejecución de su método *onCreate*. En dicho método se llevan a cabo varias fases además de la asociación de la vista correspondiente a la actividad. Se deben crear los objetos que gestionen los dos botones de respuesta así como el objeto que gestiona la opción de deslizar el dedo por la pantalla. Tras crear estos objetos, carga el fichero de audio que le corresponde e inicia su ejecución. Una vez que se ha realizado la creación de la actividad, se espera hasta que se produzca un evento. Estos eventos pueden ser la pulsación del botón de “Repetir audio”, la pulsación de uno de los botones de respuesta, o el deslizar pantalla. En caso de cualquiera de las últimas dos opciones, se

producirá un cambio de actividad, por lo que en esos casos se terminará la ejecución de la actividad actual y se lanzará la otra actividad. En el siguiente diagrama vemos la secuencia en caso de que el usuario haya pulsado una de las respuestas.

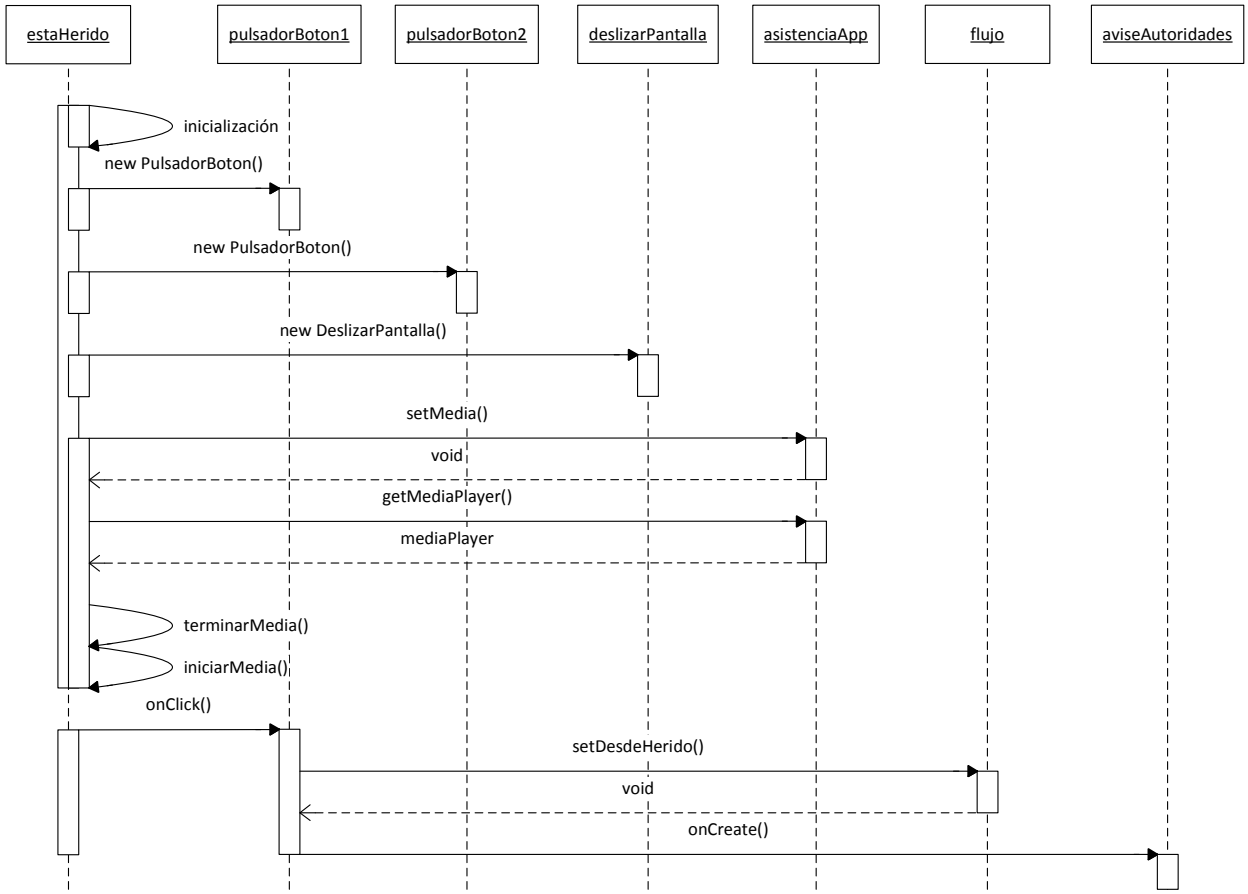


FIGURA 76: DIAGRAMA DE SECUENCIA DEL CICLO DE VIDA DE UNA ACTIVIDAD DE PREGUNTA

El segundo de los casos que podemos ver en detalle, es el de las actividades correspondientes a imágenes. Estas, en lugar de cargar dos botones, cargan una imagen o un vídeo. Vamos a centrarnos en el caso de cargar una imagen, ya que los videos se cargan de la misma manera que los archivos de audio, y utilizan los mismos métodos para iniciar y terminar su reproducción. En este caso, vamos a suponer que el usuario, una vez iniciada la pantalla, pulsa el botón de repetir audio, y luego desliza el dedo de derecha a izquierda, para pasar a la siguiente pantalla.

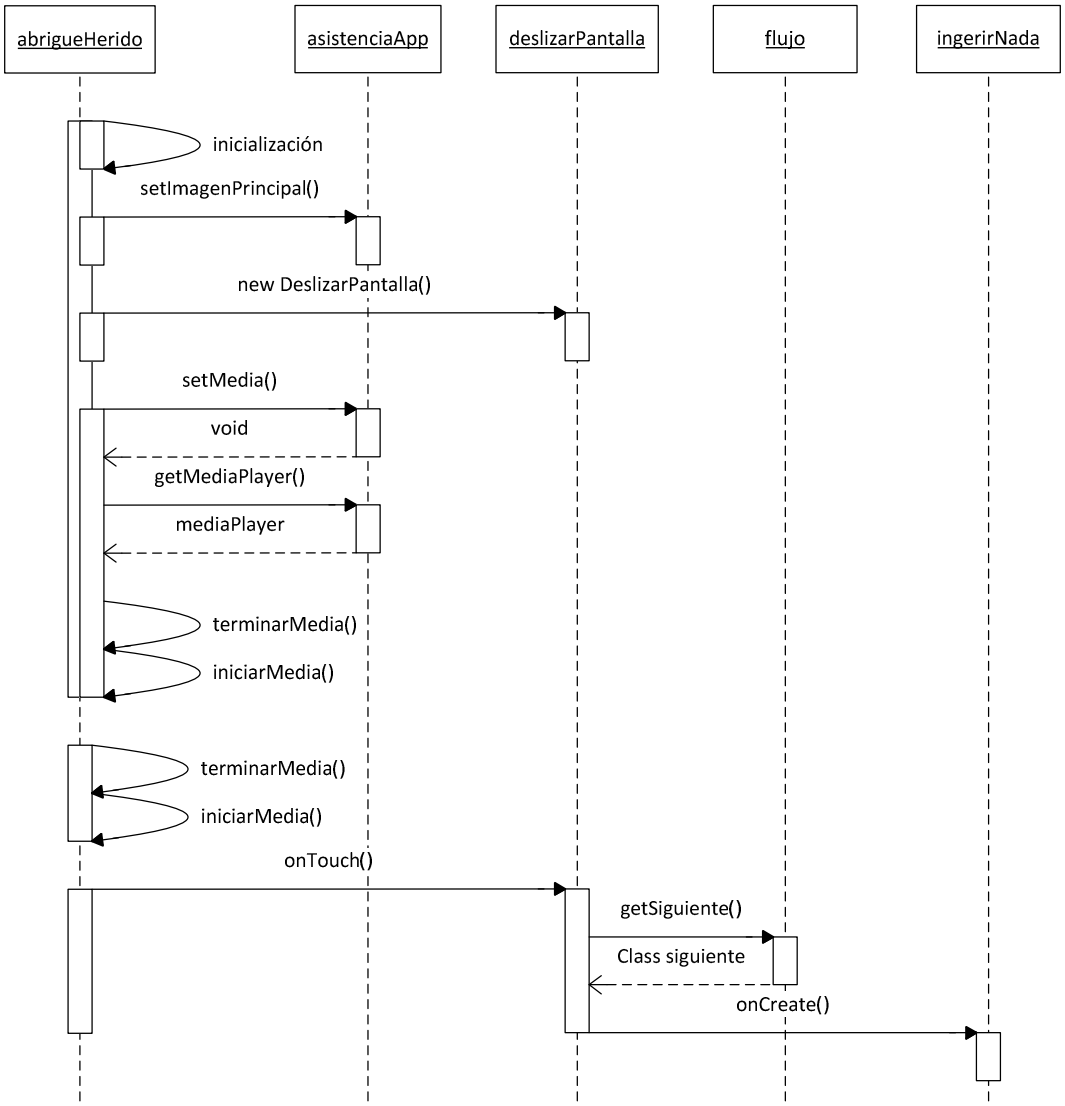


FIGURA 77: DIAGRAMA DE SECUENCIA DEL CICLO DE VIDA DE UNA ACTIVIDAD DE ACCIÓN

Como puede verse con estos dos diagramas, la creación de ambos tipos de actividades es muy similar. De hecho, todas las actividades van a tener una inicialización prácticamente idéntica a estas dos, y solo van a variar unos pocos comportamientos concretos. Entre estos comportamientos vamos a destacar el correspondiente a la carga del mapa con una llamada a los servicios de emergencias. Vamos a centrarnos en la creación del mapa y la colocación de su correspondiente marcador con la dirección. Estas acciones se llevan a cabo a la hora de inicializar la actividad, justo después de iniciar el audio. Para ver un flujo más completo de esta

pantalla, introducimos también una llamada a los servicios de emergencias por parte del usuario, mostrando el flujo completo que cabe esperar en esta actividad.

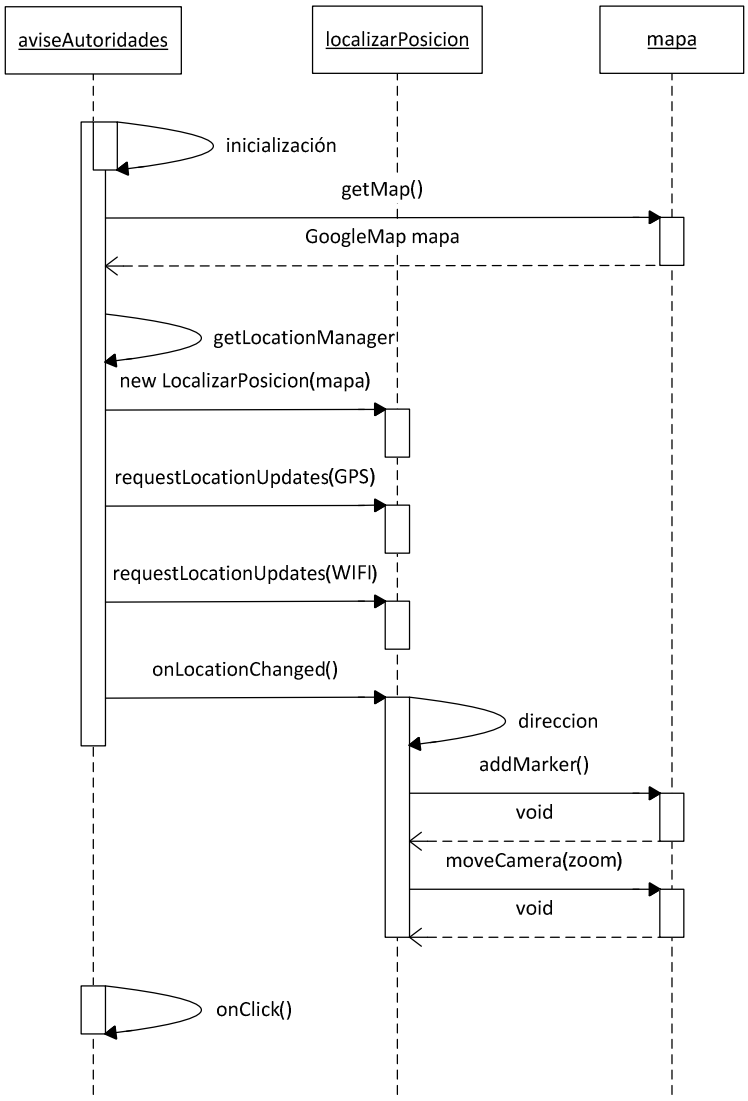


FIGURA 78: DIAGRAMA DE SECUENCIA DE LA OBTENCIÓN DE LA UBICACIÓN POR EL MAPA

Como puede verse, una vez obtenida una instancia de un mapa, se crea un objeto de tipo *LocalizarPosicion* y se le piden ubicaciones tanto por GPS como por WIFI. Cuando se obtenga una localización nueva, se lanza el evento *onLocationChanged* que hace que se obtenga la dirección de la nueva localización, y añade el marcador al mapa con esa dirección. Finalmente, se fija el zoom del mapa.

5

PLAN DE PRUEBAS

5. PLAN DE PRUEBAS

5.1. DEFINICIÓN DEL PLAN DE PRUEBAS

Para comprobar que la implementación realizada lleva a cabo toda la funcionalidad requerida, se van a diseñar un conjunto de casos de prueba. Cada caso constará de distintos apartados a comprobar que tendrán que ser validados para certificar el correcto funcionamiento de la aplicación. Para definir las pruebas se va a contar con una tabla que defina cada prueba a realizar y que registre el resultado de la misma.

Prueba 1	Nombre prueba 1	Descripción prueba 1	Resultado
Prueba 2	Nombre prueba 2	Descripción prueba 2	Resultado
Prueba n	Nombre prueba n	Descripción prueba n	Resultado

TABLA 97: FORMATO TABLA DE PLAN DE PRUEBAS

- Nombre: Nombre descriptivo corto de la prueba que se va a realizar
- Descripción: Descripción completa de la prueba que se debe realizar y en caso de ser necesario, se incluirá una descripción del resultado esperado.
- Resultado: Podrá tener dos valores posibles: Validado o No Validado.

5.2. RESULTADOS DE LAS PRUEBAS

➢ Pruebas de pantallas

Para llevar a cabo estas pruebas de forma completa es necesario seguir el siguiente flujo de acción:

1. Iniciar la aplicación y observar la primera pantalla. Validar la prueba 1.
2. Responder a la pregunta inicial con un “He visto un accidente” y observar la siguiente pantalla. Validar la prueba 2 y la prueba 3.
3. Deslizar el dedo de derecha a izquierda y una vez cambiada la pantalla hacer lo propio de izquierda a derecha. Validar la prueba 4.
4. Pulsar el botón “volver” del dispositivo. Validar la prueba 5.
5. Pulsar el botón “menú” del dispositivo. Validar la prueba 6.
6. Seleccionar la opción “S.O.S” del menú. Validar la prueba 7.
7. Pulsar de nuevo el botón “menú” del dispositivo y seleccionar la opción “Datos guardados”. Validar la prueba 8.

8. Pulsar de nuevo el botón “menú” del dispositivo y seleccionar la opción “Ayuda”. Validar la prueba 9.
9. Pulsar el botón “menú” y seleccionar la opción “S.O.S”. Una vez en el inicio, seleccionar la respuesta “He tenido un accidente”. En la siguiente pregunta, responder con un “No” y acto seguido avanzar hasta llegar a la pregunta “¿Desea guardar los datos del accidente?”. Una vez allí, responder “Sí”. Validar la prueba 10.
10. Pulsar el botón “Tomar fotos” y tomar una foto. Validar la prueba 12.
11. Pulsar el botón “Guardar”. Validar la prueba 11.
12. Pulsar el botón “Guardar” y después seleccionar el menú “Datos guardados”. En la lista de accidentes, seleccionar el accidente recién guardado y pulsar el botón “Ver fotos” y cargar la foto hecha previamente. Validar la prueba 13.

Prueba 1	Presentación pantallas de preguntas	Comprobar que en las pantallas de preguntas se muestran correctamente el texto de la pregunta, los botones con las respuestas y la flecha izquierda en caso necesario.	Validado
Prueba 2	Presentación pantallas de imágenes	Comprobar que en las pantallas de imágenes se muestran correctamente el texto descriptivo, la imagen, el botón de reproducir audio con su texto y las flechas izquierda y derecha.	Validado
Prueba 3	Cambio pantalla al pulsar respuesta	Comprobar que al pulsar los botones de respuesta se pasa a la siguiente pantalla.	Validado
Prueba 4	Cambio pantalla al deslizar	Comprobar que se pasa a la pantalla anterior si se desliza el dedo hacia la izquierda y a la pantalla siguiente si se hace lo propio hacia la derecha.	Validado
Prueba 5	Funcionamiento del botón “volver”	Comprobar que al pulsar el botón “volver” de los dispositivos Android se va a la pantalla anterior.	Validado

Prueba 6	Funcionamiento del botón “menú”	Comprobar que al pulsar el botón “menú” de los dispositivos Android se despliegan las opciones del menú.	Validado
Prueba 7	Funcionamiento menú S.O.S	Comprobar que al seleccionar la opción del menú “S.O.S” se va a la pantalla inicial de la aplicación.	Validado
Prueba 8	Funcionamiento menú Datos guardados	Comprobar que al seleccionar la opción del menú “Datos guardados” se va a la pantalla que lista los accidentes guardados.	Validado
Prueba 9	Funcionamiento menú Ayuda	Comprobar que al seleccionar la opción del menú “Ayuda” se muestran las pantallas de ayuda.	Validado
Prueba 10	Presentación pantalla de datos de accidente	Comprobar que en la pantalla correspondiente a los datos del accidente se muestran todos los campos (fecha, lugar y descripción), así como los botones correspondientes (tomar fotos y guardar en caso de estar guardando los datos, y ver fotos si se están visualizando).	Validado
Prueba 11	Presentación pantalla de datos de asegurado	Comprobar que en la pantalla correspondiente a los datos del accidente se muestran todos los campos (nombre, NIF, domicilio, teléfono, matrícula, marca y modelo, aseguradora y póliza), así como los botones correspondientes (guardar en caso de estar guardando los datos).	
Prueba 12	Tomar fotos de un accidente	Comprobar que al pulsar el botón “Tomar foto” de la pantalla correspondiente a los datos del accidente, se realizan fotos.	Validado

Prueba 13	Recuperar fotos de un accidente	Comprobar que al pulsar el botón "Ver fotos" de la pantalla correspondiente a los datos del accidente, se cargan fotos realizadas previamente.	Validado

TABLA 98: PLAN DE PRUEBAS I: PRUEBAS DE PANTALLAS

- Pruebas de flujo de pantallas

Para llevar a cabo estas pruebas de forma completa es necesario seguir el siguiente flujo de acción:

1. Ir al inicio de la aplicación (iniciándola de nuevo o pulsando el menú S.O.S) y pulsar "He visto un accidente". Avanzar hasta la pregunta "¿Está bien iluminada la zona?" y volver a retroceder hasta el inicio. Validar la prueba 1.
2. Avanzar de nuevo hasta la pregunta "¿Está bien iluminada la zona?", responder "Si" y volver. Validar la prueba 2.
3. Responder a la pregunta "¿Está bien iluminada la zona?" con un "No", avanzar hasta la pregunta "¿Hay un incendio?" y volver. Validar la prueba 3.
4. Avanzar de nuevo hasta la pregunta "¿Hay un incendio?" y una vez allí responder "Sí" y avanzar hasta la pregunta "¿Hay algún herido?". Una vez allí volver. Validar la prueba 4.
5. Responder a la pregunta "¿Hay un incendio?" con un "No", avanzar hasta la pregunta "¿Hay algún herido?" y volver. Validar la prueba 5.
6. Avanzar de nuevo hasta la pregunta "¿Hay algún herido?". Una vez allí, responder "Sí", avanzar hasta la pregunta "¿Está consciente?" y volver. Validar la prueba 6.
7. Responder ahora a la pregunta "¿Hay algún herido?" con un "No" y volver. Validar la prueba 7.
8. Avanzar de nuevo a la pregunta "¿Está consciente?". Una vez allí, responder con un "Sí" y volver. Validar la prueba 8.
9. Responder ahora a la pregunta "¿Está consciente?" con un "No", avanzar hasta la pantalla de fin y volver. Validar la prueba 9.
10. Responder de nuevo con un "Sí" a la pregunta "¿Está consciente?" y de nuevo "Sí" a la pregunta "¿Tiene heridas abiertas?" y volver un paso. Validar prueba 10.

11. Responder ahora “No” a la pregunta “¿Tiene heridas abiertas?”, avanzar hasta la pantalla de fin y volver. Validar la prueba 11.
12. Responder de nuevo “Sí” a la pregunta “¿Tiene heridas abiertas?” y de nuevo “Sí” a la pregunta “¿La hemorragia es incontrolable?”. Avanzar hasta la pantalla de fin y volver. Validar la prueba 12.
13. Responder ahora “No” a la pregunta “¿La hemorragia es incontrolable?”, avanzar hasta la pantalla de fin y volver. Validar la prueba 13.
14. Volver a la pantalla inicial mediante el menú “S.O.S” y una vez allí seleccionar “He tenido un accidente” y volver. Validar la prueba 14.
15. Volver a seleccionar “He tenido un accidente” y a la pregunta “¿Está herido?” responder con un “Sí”. Avanzar hasta el fin y volver. Validar la prueba 15.
16. Responder ahora “No” a la pregunta “¿Está herido?”, avanzar hasta la pregunta “¿Desea guardar los datos del accidente?” y volver. Validar la prueba 16.
17. Avanzar de nuevo hasta la pregunta “¿Desea guardar los datos del accidente?” y una vez allí responder “Sí”. Pulsar botón guardar en las pantallas siguientes hasta llegar al fin y volver. Validar la prueba 17.
18. Responder ahora con un “No” a la pregunta “¿Desea guardar los datos del accidente?” y volver. Validar la prueba 18.

Prueba 1	Realizar un flujo completo desde “Visto accidente” hasta “¿Bien iluminado?”	Comprobar el camino en caso de seleccionar “He visto un accidente” hasta llegar al caso “¿Bien iluminado?”. Volver hacia atrás para comprobar que al retroceder el camino es el correcto.	Validado
Prueba 2	Continuar flujo desde “¿Bien iluminado?” hasta “¿Hay incendio?” respondiendo “Sí”	Comprobar el camino desde “¿Bien iluminado?” hasta “¿Hay incendio?” respondiendo a la primera pregunta con un “Sí”. Volver hacia atrás para comprobar	Validado

		que al retroceder el camino es correcto.	
Prueba 3	Continuar flujo desde "¿Bien iluminado?" hasta "¿Hay incendio?" respondiendo "No"	Comprobar el camino desde "¿Bien iluminado?" hasta "¿Hay incendio?" respondiendo a la primera pregunta con un "No". Volver hacia atrás para comprobar que al retroceder el camino es correcto.	Validado
Prueba 4	Continuar flujo desde "¿Hay incendio?" hasta "¿Hay heridos?" respondiendo "Sí"	Comprobar el camino desde "¿Hay incendio?" hasta "¿Hay heridos?" respondiendo a la primera pregunta con un "Sí". Volver hacia atrás para comprobar que al retroceder el camino es correcto.	Validado
Prueba 5	Continuar flujo desde "¿Hay incendio?" hasta "¿Hay heridos?" respondiendo "No"	Comprobar el camino desde "¿Hay incendio?" hasta "¿Hay heridos?" respondiendo a la primera pregunta con un "No". Volver hacia atrás para comprobar que al retroceder el camino es correcto.	Validado
Prueba 6	Continuar flujo desde "¿Hay heridos?" hasta "¿Está consciente?" respondiendo "Sí"	Comprobar el camino desde "¿Hay heridos?" hasta "¿Está consciente?" respondiendo a la primera pregunta con un "Sí". Volver hacia atrás para comprobar que al retroceder el camino es correcto.	Validado
Prueba 7	Continuar flujo desde "¿Hay heridos?" hasta "Espere autoridades" respondiendo "No"	Comprobar el camino desde "¿Hay heridos?" hasta "Espere autoridades" respondiendo a la primera pregunta con un "No". Volver hacia atrás para comprobar	Validado

		que al retroceder el camino es correcto.	
Prueba 8	Continuar flujo desde "¿Está consciente?" hasta "¿Tiene Heridas?" respondiendo "Sí"	Comprobar el camino desde "¿Está consciente?" hasta "¿Tiene Heridas?" respondiendo a la primera pregunta con un "Sí". Volver hacia atrás para comprobar que al retroceder el camino es correcto.	Validado
Prueba 9	Continuar flujo desde "¿Está consciente?" hasta "Espere autoridades" respondiendo "No"	Comprobar el camino desde "¿Está consciente?" hasta "Espere autoridades" respondiendo a la primera pregunta con un "No". Volver hacia atrás para comprobar que al retroceder el camino es correcto.	Validado
Prueba 10	Continuar flujo desde "¿Tiene Heridas?" hasta "¿Hemorragia incontrolable?" respondiendo "Sí"	Comprobar el camino desde "¿Tiene Heridas?" hasta "¿Hemorragia incontrolable?" respondiendo a la primera pregunta con un "Sí". Volver hacia atrás para comprobar que al retroceder el camino es correcto.	Validado
Prueba 11	Continuar flujo desde "¿Tiene Heridas?" hasta "Espere autoridades" respondiendo "No"	Comprobar el camino desde "¿Tiene Heridas?" hasta "Espere autoridades" respondiendo a la primera pregunta con un "No". Volver hacia atrás para comprobar que al retroceder el camino es correcto.	Validado
Prueba 12	Continuar flujo desde "¿Hemorragia incontrolable?" hasta	Comprobar el camino desde "¿Hemorragia incontrolable?" hasta "Espere autoridades"	Validado

	"Espere autoridades" respondiendo "Sí"	respondiendo a la primera pregunta con un "Sí". Volver hacia atrás para comprobar que al retroceder el camino es correcto.	
Prueba 13	Continuar flujo desde "¿Hemorragia incontrolable?" hasta "Espere autoridades" respondiendo "No"	Comprobar el camino desde "¿Hemorragia incontrolable?" hasta "Espere autoridades" respondiendo a la primera pregunta con un "No". Volver hacia atrás para comprobar que al retroceder el camino es correcto.	Validado
Prueba 14	Realizar un flujo completo desde "Tenido accidente" hasta "¿Está herido?"	Comprobar el camino en caso de seleccionar "He tenido un accidente" hasta llegar al caso "¿Está herido?". Volver hacia atrás para comprobar que al retroceder el camino es el correcto.	Validado
Prueba 15	Continuar flujo desde "¿Está herido?" hasta "Fin" respondiendo "Sí"	Comprobar el camino desde "¿Está herido?" hasta "Fin" respondiendo a la primera pregunta con un "Sí". Volver hacia atrás para comprobar que al retroceder el camino es correcto.	Validado
Prueba 16	Continuar flujo desde "¿Está herido?" hasta "¿Guardar datos?" respondiendo "No"	Comprobar el camino desde "¿Está herido?" hasta "¿Guardar datos?" respondiendo a la primera pregunta con un "No". Volver hacia atrás para comprobar que al retroceder el camino es correcto.	Validado
Prueba 17	Continuar flujo desde "¿Guardar datos?" hasta "Fin" respondiendo "Sí"	Comprobar el camino desde "¿Guardar datos?" hasta "Fin" respondiendo a la primera pregunta con un "Sí". Volver hacia atrás	Validado

		para comprobar que al retroceder el camino es correcto.	
Prueba 18	Continuar flujo desde "¿Guardar datos?" hasta "Fin" respondiendo "No"	Comprobar el camino desde "¿Guardar datos?" hasta "Fin" respondiendo a la primera pregunta con un "No". Volver hacia atrás para comprobar que al retroceder el camino es correcto.	Validado

TABLA 99: PLAN DE PRUEBAS II: PRUEBAS DE FLUJO DE PANTALLAS

- Pruebas del mapa y la llamada de emergencias

Para llevar a cabo estas pruebas de forma completa es necesario seguir el siguiente flujo de acción:

1. Iniciar la aplicación y avanzar hasta llegar a la pantalla con el mapa. Validar la prueba 1, la prueba 3 y la prueba 6.
2. Pulsar el botón de llamada al 112. Validar la prueba 2.
3. Acercar la vista del mapa y luego volver a alejarla. Validar la prueba 4.
4. Mover la vista del mapa. Validar la prueba 5.
5. Pulsar sobre el indicador de posición y comprobar la dirección. Validar la prueba 7.

Prueba 1	Mostrado botón 112	Comprobar que se muestra correctamente el botón "112".	Validado
Prueba 2	Llamada al pulsar botón 112	Comprobar que al pulsar el botón "112" se realiza una llamada al servicio de emergencias.	Validado
Prueba 3	Mostrado mapa	Comprobar que se carga correctamente el mapa de Google.	Validado
Prueba 4	Funcionamiento botones "acercar" y "alejar" del mapa	Comprobar que al pulsar los botones de "acercar" y "alejar" del mapa, este responde correctamente.	Validado
Prueba	Movimiento del mapa	Comprobar que se puede deslizar	Validado

5		el dedo por el mapa moviendo la vista del mismo.	
Prueba 6	Mostrada señalización del lugar	Comprobar que se muestra el marcador que indica la posición actual.	Validado
Prueba 7	Mostrar dirección al pulsar señalización	Comprobar que al pulsar en el marcador del mapa, se muestra la dirección actual.	Validado

TABLA 100: PLAN DE PRUEBAS III: PRUEBAS DEL MAPA Y LLAMADA DE EMERGENCIAS

- Pruebas de los reproductores de sonido y video

Para llevar a cabo estas pruebas de forma completa es necesario seguir el siguiente flujo de acción:

1. Iniciar la aplicación y avanzar hasta una pantalla de acción asegurándose de tener el audio a un volumen perceptible. Validar la prueba 1.
2. Antes de que finalice el audio, cambiar de pantalla. Validar la prueba 2.
3. Volver a una pantalla de acción y pulsar el botón de repetir audio al finalizar la reproducción y antes de que termine. Validar la prueba 3.
4. Avanzar hasta una pantalla que contenga un video. Validar la prueba 4.
5. Cambiar de pantalla mientras se está reproduciendo el video. Validar la prueba 5.
6. Pulsar sobre el video y controlar la reproducción con los botones de control. Validar la prueba 6.

Prueba 1	Reproducción de audio al iniciar pantalla	Comprobar que al iniciar una pantalla se reproduce correctamente el audio correspondiente.	Validado
Prueba 2	Cortar reproducción de audio al cambiar pantalla	Comprobar que al cambiar de pantalla mientras está reproduciendo un audio, este termina su reproducción.	Validado
Prueba 3	Reiniciar reproducción de audio al pulsar botón de	Comprobar que al pulsar el botón de “Repetir Audio”, se vuelve a iniciar la reproducción, aunque aún no haya	Validado

	repetir	terminado.	
Prueba 4	Reproducción de video al iniciar pantalla	Comprobar que al iniciar una pantalla se reproduce correctamente el video correspondiente.	Validado
Prueba 5	Cortar reproducción de video al cambiar de pantalla	Comprobar que al cambiar de pantalla mientras está reproduciendo un video, este termina su reproducción.	Validado
Prueba 6	Mostrar botones de control de video	Comprobar que al pulsar en el video, se muestran los botones para controlar la reproducción (reproducir/pausar, avanzar y retroceder).	Validado

TABLA 101: PLAN DE PRUEBAS IV: PRUEBAS DE REPRODUCTORES DE AUDIO Y VÍDEO

- Pruebas de la base de datos

Para llevar a cabo estas pruebas de forma completa es necesario seguir el siguiente flujo de acción:

1. Iniciar la aplicación y avanzar hasta la pantalla de introducir datos del accidente. Una vez allí, introducir los datos y pulsar guardar. A continuación, no introducir ningún dato de asegurado y guardar.
2. Acceder al menú “Datos Guardados”. Validar prueba 3.
3. Seleccionar el accidente que se acaba de guardar. Validar pruebas 1 y 4.
4. Volver al inicio de la aplicación y avanzar de nuevo hasta la pantalla de introducir datos del accidente. Insertar datos y guardar. A continuación, introducir datos del asegurado y guardar.
5. Acceder al menú “Datos Guardados” de nuevo y seleccionar el accidente que se acaba de guardar. Validar las pruebas 2 y 5.

Prueba 1	Guardar datos introducidos de accidente	Comprobar que al introducir datos de accidente únicamente, estos se guardan correctamente en la base de datos.	Validado
Prueba 2	Guardar datos introducidos de accidente y	Comprobar que al introducir datos de accidente y asegurado, ambos se guardan correctamente en la base de datos.	Validado

	asegurado		
Prueba 3	Cargar lista de accidentes guardados	Comprobar que al cargar los accidentes guardados, se listan todos los que hay almacenados, cada uno con su botón correspondiente.	Validado
Prueba 4	Cargar datos de accidente	Comprobar que al cargar los datos de un accidente únicamente, estos se muestran correctamente.	Validado
Prueba 5	Cargar datos de accidente y asegurado	Comprobar que al cargar los datos de un accidente y un asegurado, ambos se muestran correctamente.	Validado

TABLA 102: PLAN DE PRUEBAS V: PRUEBAS DE LA BASE DE DATOS

6

CONCLUSIONES

6. CONCLUSIONES

6.1. CONCLUSIONES DEL DESARROLLO

La realización de este Proyecto de Fin de Carrera ha supuesto llevar a cabo el desarrollo de una aplicación completa, partiendo desde cero. Este tipo de desarrollos hace que se lleven a cabo todos los pasos del desarrollo software, desde el estudio inicial hasta la implementación final, pasando por el análisis de todos los requisitos y posibles soluciones. Esto hace que se haya conseguido visualizar completamente la complejidad de desarrollar un producto software que pueda ser consumido por los usuarios.

Centrándonos en las partes más propias del desarrollo en sí, que se corresponden con el análisis y el diseño, se ha podido observar cómo realizar un diseño de una aplicación móvil no resulta tan sencillo como puede parecer en un primer momento. La necesidad de diseñar una interfaz para distintas resoluciones de pantalla conlleva una complejidad bastante alta a la hora de llevar a cabo un diseño de interfaz que sea totalmente útil para las distintas versiones de dispositivos que se tienen actualmente en el mercado. Es por ello que se terminó por optar por un diseño simple, especialmente diseñado para móviles en lugar de para tabletas, pero intentando causar un impacto lo menor posible en todos los dispositivos.

Hay que destacar también que a la hora de publicar una aplicación móvil, hay que realizar un trabajo extra para entender las posibilidades de publicación. Pese a que hay plataformas de distribución oficiales para las aplicaciones móviles, es posible también publicarlas en mercados de terceros, haciendo que la aplicación pueda llegar a más usuarios. Además, hay que tener en cuenta que el uso de un mapa de Google, conlleva la necesidad de dar de alta la aplicación en los servicios de localización de Google, por lo que ha sido necesario familiarizarse con las herramientas que te proporciona la propia compañía para registrar sus servicios en tu aplicación.

6.2. APORTACIONES PERSONALES DEL PROYECTO

En cuanto a las aportaciones a nivel personal, este proyecto me ha dado la oportunidad de investigar la posibilidad de desarrollar aplicaciones móviles con la tecnología Android. Esto es algo que valoro sobremanera, ya que el mercado de este tipo de aplicaciones está experimentando un desarrollo muy importante en los últimos

años, y cada vez son más los dispositivos que hay en el mercado, y más potentes. Pensando en este tipo de tecnologías como las más punteras que pueden dominar el desarrollo software de los próximos años, considero que ha sido una gran oportunidad el poder enfrentarme a un proyecto de estas características.

Además, he podido vivir más de cerca lo que supone un desarrollo completo, pudiendo ver con detalle las distintas fases de desarrollo, de manera que he podido aprender muchas cosas importantes de cara a tomar decisiones laborales en el futuro, cuando me enfrente a posibles tomas de decisiones sobre cómo llevar a cabo un parte de un desarrollo. Incluso he podido ver en qué aspectos del desarrollo me encuentro más cómodo y en cuales necesito hacer un poco más de hincapié de cara al futuro.

Personalmente me gustaría recalcar que no todos los tipos de aplicaciones resultan igual de gratificantes de desarrollar. Poder ofrecer a todos los usuarios de dispositivos Android una aplicación que pueda resultarles útil en un momento de peligro supone una gran satisfacción personal. El hecho de realizar un proyecto que pueda ayudar a las personas a superar una situación complicada de la mejor manera posible, pudiendo incluso preparar el terreno para que la actuación de los servicios de emergencias (si es necesaria) sea más efectiva, para mí supone que el Proyecto de Fin de Carrera no ha sido sólo un mero trámite académico, sino algo realmente útil para los demás. Y eso creo que es un aspecto que le da al proyecto una dosis extra de importancia.

6.3. AMPLIACIONES FUTURAS

Para finalizar este proyecto, me gustaría también nombrar aquellos aspectos que se han quedado fuera del desarrollo y que sin embargo, creo que pueden ser útiles de cara a posibles mejoras futuras a realizar en la aplicación.

Como ya he nombrado, el diseño de cara a las tabletas es algo que se puede revisar e incluso incorporar posibilidades adicionales. Una pantalla más grande ofrece posibilidades que quedan fuera del alcance de los *Smartphone*. Pero no solo se puede mejorar a nivel de diseño. Analizando las aplicaciones de la competencia, se ha visto que todas las compañías aseguradoras tienen su propia aplicación móvil, que sin abarcar los pasos que abarca este proyecto, proporcionan algunas funcionalidades que podrían complementar este desarrollo. Entre estas, me gustaría destacar la posibilidad de encontrar el centro médico más cercano y proporcionar la ruta más rápida para llegar

desde el lugar del accidente. Añadir esta funcionalidad daría una posibilidad extra para afrontar la situación de encontrarse con heridos leves.

Fuera ya un poco de la temática de asistencia a accidentes que es el centro de la aplicación desarrollada, un posible complemento que podría ser muy útil también a los usuarios sería el relativo a problemas mecánicos típicos. Saber cómo detectarlos y cómo solucionarlos o minimizarlos, puede hacer que se disminuyan los riesgos de tener un accidente. Este tipo de mejora entraría más en el propósito de prevención de accidentes que en el de asistencia de accidentes, pero podría ser un complemento ideal a la aplicación.

ANEXO I: MANUAL DE USUARIO

Para poder hacer uso de la aplicación, es necesario acceder a la plataforma de distribución de aplicaciones de Android, Google Play, y buscar la aplicación "Asistencia en carretera". Una vez descargada e instalada la aplicación, lo primero que se verá será su icono.

FIGURA 79: ICONO DE LA APLICACIÓN

Al acceder por primera vez a la aplicación, se mostrará la pantalla inicial, correspondiente a la pregunta "¿Qué tipo de incidencia ha ocurrido?". A partir de ese momento, caben varias posibilidades. La primera, consiste en pulsar una de las dos respuestas posibles. Estás nos llevarán al comienzo de los dos flujos principales de acciones que debe llevar a cabo el usuario.

FIGURA 80: PANTALLA INICIAL DE LA APLICACIÓN

La segunda opción, consiste en seleccionar cualquier de las opciones del menú. Para ello, basta con pulsar el botón "menú" incluido en todos los dispositivos Android, y seleccionar la opción que queramos.

FIGURA 81: VISTA DEL MENÚ DE LA APLICACIÓN

Para moverse entre las distintas pantallas, para ver los pasos que debe seguir el usuario, hay tres opciones posibles. La primera de ellas consiste en deslizar el dedo por la pantalla, de derecha a izquierda o de izquierda a derecha. En el primer caso, lo que estaremos haciendo será retroceder hacia la pantalla anterior, mientras que en el segundo lo estaremos haciendo hacia la pantalla siguiente. No siempre va a ser posible llevar a cabo estos desplazamientos. Cada pantalla podrá incorporar una flecha señalando hacia la izquierda para indicar que podemos retroceder y/o una flecha señalando a la derecha para indicar que podemos avanzar. La segunda opción para movernos entre pantallas permite únicamente retroceder mediante la tecla “volver” incluida en los dispositivos Android. Finalmente, en el caso de que estemos en una pantalla con una pregunta, para poder avanzar habrá que seleccionar una de las respuestas posibles.

FIGURA 82: VISTA DE LOS DESPLAZAMIENTOS ENTRE PANTALLAS DE LA APLICACIÓN

La gran mayoría de las pantallas contienen un audio que será reproducido al entrar en ellas. Este audio podrá volver a reproducirse pulsando el botón correspondiente. Las pantallas que no cuentan con audio, contarán con un vídeo explicativo de la acción que ha de realizar el usuario. Estos vídeos se comenzarán a ejecutar al entrar la pantalla que lo contiene, y contarán con unos botones para controlar su reproducción. Estos botones se mostrarán durante unos segundos. Para volver a mostrarlos, bastará con pulsar sobre el vídeo.

FIGURA 83: VISTA DE LOS CONTROLES DEL AUDIO Y VÍDEO DE LA APLICACIÓN

En cualquier camino que siga el usuario, antes o después según la gravedad de la situación, se le mostrará una pantalla especial en la cual aparecerá un mapa con un marcador que indique la posición en la que se encuentra y un botón para contactar con los servicios de emergencias directamente. El marcador inicialmente aparecerá sin

informar de la calle. En caso de querer saber la dirección en la que se encuentra, se deberá pulsar el marcador del mapa. Además, se podrá alejar o acercar la vista del mapa mediante los botones del zoom.

FIGURA 84: VISTA DEL MAPA DE LA APLICACIÓN

En el caso de que el usuario haya sufrido un accidente, y no se encuentre herido, al final de las acciones que tiene que llevar a cabo para asegurarse de ponerse en peligro ni poner en peligro a otros, se le ofrecerá la opción de guardar información relativa al accidente. Esta información se va a presentar en dos partes. Por un lado, se pedirá la información básica relativa al accidente. Esta información consta de la fecha y la hora, el lugar en el que se ha producido y una descripción del mismo. Además, se permitirá al usuario tomar fotos del accidente si lo cree necesario. Estas fotos irán nombradas con la fecha y la hora del accidente, seguidas de un número de foto. Por defecto, el dato de fecha y hora vendrá relleno con la fecha y la hora actuales, y el lugar vendrá relleno con la ubicación dada por el mapa. Ambos datos podrán ser modificados por el usuario en caso de creerlo conveniente. Tras esta información, el usuario podrá indicar si el accidente se ha llevado a cabo con un contrario, en cuyo caso se pasará a mostrar la información relativa a los datos del contrario. Esta información incluirá datos básicos que pueden ser necesarios para el usuario con posterioridad al accidente, y consisten en el nombre y apellidos del contrario, su NIF, dirección, teléfono, matrícula del coche, marca y modelo, aseguradora y número de póliza.

FIGURA 85: VISTA DE LOS DATOS DEL ACCIDENTE Y DEL ASEGURADO DE LA APLICACIÓN

Una vez guardados los datos, estos podrán ser recuperados mediante la opción "Datos guardados" del menú. Al seleccionar esta opción, se mostrará una pantalla con todos los accidentes almacenados y al pulsar en uno de ellos, se cargará la información asociada. Esta información se mostrará en dos pantallas similares a las disponibles para introducir los datos, salvo que sin los botones para guardarlos, y en lugar de un botón para tomar fotos en los datos del accidente, aparecerá un botón para ver las fotos que haya realizado previamente. Hay que recordar que las fotos correspondientes al accidente vendrán nombradas con la fecha y hora del mismo.

FIGURA 86: VISTA DE LA LISTA DE ACCIDENTES GUARDADOS DE LA APLICACIÓN

Las opciones del menú pueden ser seleccionadas estando en cualquier pantalla de la aplicación. Como ya se ha mencionado, la opción "Datos guardados" lleva a la pantalla para visualizar los datos de accidentes pasados que hayan sido almacenados. La opción

“S.O.S” dirige directamente a la pantalla inicial de la aplicación, correspondiente a la pregunta “¿Qué tipo de incidencia ha ocurrido?”. Finalmente, la opción “Ayuda”, mostrará un pequeño resumen de este manual para poder consultarlo en cualquier momento en caso de necesitar saber cómo utilizar la aplicación, e información sobre el proyecto.

ANEXO II: ÍNDICE DE FIGURAS

ANEXO III: ÍNDICE DE TABLAS

ANEXO IV: REFERENCIAS

Ableson, W. Frank (2010). Android: guía para desarrolladores. Madrid: Anaya Multimedia.

Android. Android Developers. Recuperado el 1 de Febrero de 2013, de http://developer.android.com/guide/components/index.html.

Boehn, B. W (Mayo 1988). A espiral model of software development and enhancement. Computer.

Burnette, Ed (2011). Android. Madrid: Anaya Multimedia

Canós, José H., Letelier, Patricio y Penadés, Mª Carmen (2003). Metodologías ágiles en el desarrollo de software. Universidad Politécnica de Valencia.

D.G.T. Las principales cifras de la siniestralidad vial. Recuperado el 16 de Abril de 2013 de http://www.dgt.es/portal/es/seguridad_vial/estadistica/publicaciones/princip_cifras_siniestral/.

Fuggetta, Alfonso (2000). Software process: A roadmap. Proceedings of the Conference on The Future of Software Engineering, 25-34.

Google. Aplicaciones de Android en Google Play. Recuperado el 8 de Julio de 2013 de https://play.google.com/store/apps.

Holzer, Adrian y Ondrus, Jan (2009). Trends in Mobile Application Development. Middleware, Operating Systems, and Applications.

Identidadgeek. Los dueños de smartphones por grupos de edad. Recuperado el 6 de Abril de 2013 de http://identidadgeek.com/los-duenos-de-smartphones-por-grupos-de-edad/2011/11/.

Kangas, Eeva y Kinnunen, Timo (Julio 2005). Applying User-Centered Design to Mobile Application Development. Communications of the ACM 2005, Vol. 48.

Köning-Ries, Birgitta (2009). Challenges in Mobile Application Development. IT-Information Technology. Universidad de Jena.

Race. Informe RACE Eurotest 2013 sobre primerios auxilios. Recuperado el 17 de Abril de 2013 de http://www.race.es/seguridad-vial/informes.

Taringa. La historia de los smartphones. Recuperado el 6 de Abril de 2013 de http://www.taringa.net/posts/info/13875247/La-historia-de-los-smartphones.html.

Wassernan, Anthony I. (2010). Software Engineering Issues for Mobile Application Development. FoSER 2010.